



Cambricon BANG C Developer Guide

Release 4.0.2

Cambricon@155chb

Aug 15, 2022



Table of Contents

Table of Contents	i
List of Figures	1
List of Tables	2
1 Copyright	4
2 Preface	6
2.1 Version	6
2.2 Update History	8
3 Built-in Functions	15
3.1 1D Memcpy Functions	15
3.1.1 __bang_move	15
3.1.2 __memcpy	16
3.1.3 __memcpy_async	20
3.2 1D Memset Functions	23
3.2.1 __bang_write_value	23
3.2.2 __bang_write_zero	24
3.2.3 __gdramset	25
3.2.4 __ldramset	26
3.2.5 __memset_nram	27
3.2.6 __sramset	28
3.3 2D Memcpy Functions	29
3.3.1 __bang_move	29
3.3.2 __memcpy	30
3.3.3 __memcpy_async	37
3.4 3D Memcpy Functions	43
3.4.1 __bang_move	43
3.4.2 __memcpy	45
3.4.3 __memcpy_async	48
3.5 3D Memset Functions	50
3.5.1 __bang_write_value	50
3.5.2 __gdramset	52
3.5.3 __ldramset	54
3.5.4 __nramset	56
3.5.5 __sramset	58
3.6 Artificial Intelligence Functions	60

3.6.1	__bang_avgpool	60
3.6.2	__bang_avgpool_bp	63
3.6.3	__bang_bexpand	65
3.6.4	__bang_breduce	67
3.6.5	__bang_conv	68
3.6.6	__bang_conv_partial	112
3.6.7	__bang_conv_partial_tf32	140
3.6.8	__bang_conv_sparse	142
3.6.9	__bang_conv_tf32	149
3.6.10	__bang_maxpool	151
3.6.11	__bang_maxpool_bp	155
3.6.12	__bang_maxpool_index	157
3.6.13	__bang_maxpool_value_index	161
3.6.14	__bang_minpool	164
3.6.15	__bang_minpool_index	167
3.6.16	__bang_minpool_value_index	170
3.6.17	__bang_mlp	172
3.6.18	__bang_reshape_filter	179
3.6.19	__bang_reshape_nchw2nhwc	181
3.6.20	__bang_reshape_nhwc2nchw	183
3.6.21	__bang_ssparse_filter_index	185
3.6.22	__bang_ssparse_filter_sparse_index	187
3.6.23	__bang_ssparse_filter_union	190
3.6.24	__bang_ssparse_filter_union_index	192
3.6.25	__bang_smpool	195
3.6.26	__bang_unpool	197
3.7	Atomic Functions	200
3.7.1	__bang_atomic_add	200
3.7.2	__bang_atomic_and	202
3.7.3	__bang_atomic_cas	204
3.7.4	__bang_atomic_dec	205
3.7.5	__bang_atomic_exch	207
3.7.6	__bang_atomic_inc	209
3.7.7	__bang_atomic_max	211
3.7.8	__bang_atomic_min	213
3.7.9	__bang_atomic_or	215
3.7.10	__bang_atomic_xor	217
3.8	Atomic Reduce Functions	218
3.8.1	__bang_atomic_reduce_add	218
3.8.2	__bang_atomic_reduce_and	220
3.8.3	__bang_atomic_reduce_cas	221
3.8.4	__bang_atomic_reduce_dec	222
3.8.5	__bang_atomic_reduce_exch	223
3.8.6	__bang_atomic_reduce_inc	224
3.8.7	__bang_atomic_reduce_max	226
3.8.8	__bang_atomic_reduce_min	227
3.8.9	__bang_atomic_reduce_or	229

3.8.10	<code>__bang_atomic_reduce_xor</code>	230
3.9	Control Flow and Debugging Functions	231
3.9.1	<code>__assert</code>	231
3.9.2	<code>__bang_printf</code>	232
3.9.3	<code>__is_ipu</code>	233
3.9.4	<code>__is_mpu</code>	233
3.9.5	<code>__is_nram</code>	234
3.9.6	<code>__is_sram</code>	234
3.9.7	<code>__is_wram</code>	235
3.10	Discrete Atomic Functions	235
3.10.1	<code>__bang_discrete_atomic_add</code>	235
3.10.2	<code>__bang_discrete_atomic_and</code>	241
3.10.3	<code>__bang_discrete_atomic_cas</code>	243
3.10.4	<code>__bang_discrete_atomic_dec</code>	250
3.10.5	<code>__bang_discrete_atomic_exch</code>	254
3.10.6	<code>__bang_discrete_atomic_inc</code>	259
3.10.7	<code>__bang_discrete_atomic_max</code>	264
3.10.8	<code>__bang_discrete_atomic_min</code>	271
3.10.9	<code>__bang_discrete_atomic_or</code>	278
3.10.10	<code>__bang_discrete_atomic_xor</code>	281
3.11	Scalar Type Conversion Functions	283
3.11.1	<code>__bfloat162char</code>	283
3.11.2	<code>__bfloat162char_dn</code>	284
3.11.3	<code>__bfloat162char_oz</code>	284
3.11.4	<code>__bfloat162char_rd</code>	285
3.11.5	<code>__bfloat162char_rm</code>	285
3.11.6	<code>__bfloat162char_rn</code>	286
3.11.7	<code>__bfloat162char_tz</code>	286
3.11.8	<code>__bfloat162char_up</code>	287
3.11.9	<code>__bfloat162float</code>	287
3.11.10	<code>__bfloat162float_dn</code>	288
3.11.11	<code>__bfloat162float_oz</code>	288
3.11.12	<code>__bfloat162float_rd</code>	289
3.11.13	<code>__bfloat162float_rm</code>	289
3.11.14	<code>__bfloat162float_rn</code>	290
3.11.15	<code>__bfloat162float_tz</code>	290
3.11.16	<code>__bfloat162float_up</code>	291
3.11.17	<code>__bfloat162half</code>	291
3.11.18	<code>__bfloat162half_dn</code>	292
3.11.19	<code>__bfloat162half_oz</code>	293
3.11.20	<code>__bfloat162half_rd</code>	293
3.11.21	<code>__bfloat162half_rm</code>	294
3.11.22	<code>__bfloat162half_rn</code>	294
3.11.23	<code>__bfloat162half_tz</code>	295
3.11.24	<code>__bfloat162half_up</code>	295
3.11.25	<code>__bfloat162int</code>	296
3.11.26	<code>__bfloat162int_dn</code>	296

3.11.27	<u>__bfloat162int_oz</u>	297
3.11.28	<u>__bfloat162int_rd</u>	297
3.11.29	<u>__bfloat162int_rm</u>	298
3.11.30	<u>__bfloat162int_rn</u>	298
3.11.31	<u>__bfloat162int_tz</u>	299
3.11.32	<u>__bfloat162int_up</u>	299
3.11.33	<u>__bfloat162short</u>	300
3.11.34	<u>__bfloat162short_dn</u>	300
3.11.35	<u>__bfloat162short_oz</u>	301
3.11.36	<u>__bfloat162short_rd</u>	301
3.11.37	<u>__bfloat162short_rm</u>	302
3.11.38	<u>__bfloat162short_rn</u>	302
3.11.39	<u>__bfloat162short_tz</u>	303
3.11.40	<u>__bfloat162short_up</u>	303
3.11.41	<u>__bfloat162tf32</u>	304
3.11.42	<u>__bfloat162tf32_dn</u>	305
3.11.43	<u>__bfloat162tf32_oz</u>	305
3.11.44	<u>__bfloat162tf32_rd</u>	306
3.11.45	<u>__bfloat162tf32_rm</u>	306
3.11.46	<u>__bfloat162tf32_rn</u>	307
3.11.47	<u>__bfloat162tf32_tz</u>	307
3.11.48	<u>__bfloat162tf32_up</u>	308
3.11.49	<u>__char2bfloat16</u>	308
3.11.50	<u>__char2float</u>	309
3.11.51	<u>__char2half</u>	309
3.11.52	<u>__char2tf32</u>	310
3.11.53	<u>__float2bfloat16</u>	310
3.11.54	<u>__float2bfloat16_dn</u>	311
3.11.55	<u>__float2bfloat16_oz</u>	312
3.11.56	<u>__float2bfloat16_rd</u>	312
3.11.57	<u>__float2bfloat16_rm</u>	313
3.11.58	<u>__float2bfloat16_rn</u>	313
3.11.59	<u>__float2bfloat16_tz</u>	314
3.11.60	<u>__float2bfloat16_up</u>	314
3.11.61	<u>__float2char</u>	315
3.11.62	<u>__float2char_dn</u>	315
3.11.63	<u>__float2char_oz</u>	316
3.11.64	<u>__float2char_rd</u>	316
3.11.65	<u>__float2char_rm</u>	317
3.11.66	<u>__float2char_rn</u>	317
3.11.67	<u>__float2char_tz</u>	318
3.11.68	<u>__float2char_up</u>	318
3.11.69	<u>__float2half</u>	319
3.11.70	<u>__float2half_dn</u>	319
3.11.71	<u>__float2half_oz</u>	320
3.11.72	<u>__float2half_rd</u>	320
3.11.73	<u>__float2half_rm</u>	321

3.11.74 __float2half_rn	321
3.11.75 __float2half_tz	322
3.11.76 __float2half_up	322
3.11.77 __float2int	323
3.11.78 __float2int_dn	323
3.11.79 __float2int_oz	324
3.11.80 __float2int_rd	324
3.11.81 __float2int_rm	325
3.11.82 __float2int_rn	325
3.11.83 __float2int_tz	326
3.11.84 __float2int_up	327
3.11.85 __float2short	327
3.11.86 __float2short_dn	328
3.11.87 __float2short_oz	328
3.11.88 __float2short_rd	329
3.11.89 __float2short_rm	329
3.11.90 __float2short_rn	330
3.11.91 __float2short_tz	330
3.11.92 __float2short_up	331
3.11.93 __float2tf32	331
3.11.94 __float2tf32_dn	332
3.11.95 __float2tf32_oz	332
3.11.96 __float2tf32_rd	333
3.11.97 __float2tf32_rm	333
3.11.98 __float2tf32_rn	334
3.11.99 __float2tf32_tz	334
3.11.100 __float2tf32_up	335
3.11.101 __float2uchar	335
3.11.102 __float2uchar_dn	336
3.11.103 __float2uchar_oz	337
3.11.104 __float2uchar_rd	337
3.11.105 __float2uchar_rm	338
3.11.106 __float2uchar_rn	338
3.11.107 __float2uchar_tz	339
3.11.108 __float2uchar_up	339
3.11.109 __float2uint	340
3.11.110 __float2uint_dn	340
3.11.111 __float2uint_oz	341
3.11.112 __float2uint_rd	341
3.11.113 __float2uint_rm	342
3.11.114 __float2uint_rn	342
3.11.115 __float2uint_tz	343
3.11.116 __float2uint_up	343
3.11.117 __float2ushort	344
3.11.118 __float2ushort_dn	344
3.11.119 __float2ushort_oz	345
3.11.120 __float2ushort_rd	345

3.11.121 __float2ushort_rm	346
3.11.122 __float2ushort_rn	346
3.11.123 __float2ushort_tz	347
3.11.124 __float2ushort_up	347
3.11.125 __half2bfloat16	348
3.11.126 __half2bfloat16_dn	349
3.11.127 __half2bfloat16_oz	349
3.11.128 __half2bfloat16_rd	350
3.11.129 __half2bfloat16_rm	350
3.11.130 __half2bfloat16_rn	351
3.11.131 __half2bfloat16_tz	351
3.11.132 __half2bfloat16_up	352
3.11.133 __half2char	352
3.11.134 __half2char_dn	353
3.11.135 __half2char_oz	353
3.11.136 __half2char_rd	354
3.11.137 __half2char_rm	354
3.11.138 __half2char_rn	355
3.11.139 __half2char_tz	355
3.11.140 __half2char_up	356
3.11.141 __half2float	356
3.11.142 __half2float_dn	357
3.11.143 __half2float_oz	357
3.11.144 __half2float_rd	358
3.11.145 __half2float_rm	358
3.11.146 __half2float_rn	359
3.11.147 __half2float_tz	359
3.11.148 __half2float_up	360
3.11.149 __half2int	360
3.11.150 __half2int_dn	361
3.11.151 __half2int_oz	361
3.11.152 __half2int_rd	362
3.11.153 __half2int_rm	362
3.11.154 __half2int_rn	363
3.11.155 __half2int_tz	363
3.11.156 __half2int_up	364
3.11.157 __half2short	365
3.11.158 __half2short_dn	365
3.11.159 __half2short_oz	366
3.11.160 __half2short_rd	366
3.11.161 __half2short_rm	367
3.11.162 __half2short_rn	367
3.11.163 __half2short_tz	368
3.11.164 __half2short_up	368
3.11.165 __half2tf32	369
3.11.166 __half2tf32_dn	369
3.11.167 __half2tf32_oz	370

3.11.168 __half2tf32_rd	370
3.11.169 __half2tf32_rm	371
3.11.170 __half2tf32_rn	371
3.11.171 __half2tf32_tz	372
3.11.172 __half2tf32_up	372
3.11.173 __half2uchar	373
3.11.174 __half2uchar_dn	374
3.11.175 __half2uchar_oz	374
3.11.176 __half2uchar_rd	375
3.11.177 __half2uchar_rm	375
3.11.178 __half2uchar_rn	376
3.11.179 __half2uchar_tz	376
3.11.180 __half2uchar_up	377
3.11.181 __half2uint	377
3.11.182 __half2uint_dn	378
3.11.183 __half2uint_oz	378
3.11.184 __half2uint_rd	379
3.11.185 __half2uint_rm	379
3.11.186 __half2uint_rn	380
3.11.187 __half2uint_tz	380
3.11.188 __half2uint_up	381
3.11.189 __half2ushort	381
3.11.190 __half2ushort_dn	382
3.11.191 __half2ushort_oz	382
3.11.192 __half2ushort_rd	383
3.11.193 __half2ushort_rm	383
3.11.194 __half2ushort_rn	384
3.11.195 __half2ushort_tz	384
3.11.196 __half2ushort_up	385
3.11.197 __int2bfloat16	385
3.11.198 __int2bfloat16_dn	386
3.11.199 __int2bfloat16_oz	387
3.11.200 __int2bfloat16_rd	387
3.11.201 __int2bfloat16_rm	388
3.11.202 __int2bfloat16_rn	388
3.11.203 __int2bfloat16_tz	389
3.11.204 __int2bfloat16_up	389
3.11.205 __int2float	390
3.11.206 __int2float_dn	390
3.11.207 __int2float_oz	391
3.11.208 __int2float_rd	391
3.11.209 __int2float_rm	392
3.11.210 __int2float_rn	392
3.11.211 __int2float_tz	393
3.11.212 __int2float_up	393
3.11.213 __int2half	394
3.11.214 __int2half_dn	394

3.11.215 __int2half_oz	395
3.11.216 __int2half_rd	395
3.11.217 __int2half_rm	396
3.11.218 __int2half_rn	396
3.11.219 __int2half_tz	397
3.11.220 __int2half_up	397
3.11.221 __int2tf32	398
3.11.222 __int2tf32_dn	399
3.11.223 __int2tf32_oz	399
3.11.224 __int2tf32_rd	400
3.11.225 __int2tf32_rm	400
3.11.226 __int2tf32_rn	401
3.11.227 __int2tf32_tz	401
3.11.228 __int2tf32_up	402
3.11.229 __popcnt	402
3.11.230 __short2bfloat16	403
3.11.231 __short2bfloat16_dn	403
3.11.232 __short2bfloat16_oz	404
3.11.233 __short2bfloat16_rd	404
3.11.234 __short2bfloat16_rm	405
3.11.235 __short2bfloat16_rn	405
3.11.236 __short2bfloat16_tz	406
3.11.237 __short2bfloat16_up	406
3.11.238 __short2fdat	407
3.11.239 __short2half	407
3.11.240 __short2half_dn	408
3.11.241 __short2half_oz	408
3.11.242 __short2half_rd	409
3.11.243 __short2half_rm	409
3.11.244 __short2half_rn	410
3.11.245 __short2half_tz	410
3.11.246 __short2half_up	411
3.11.247 __short2tf32	411
3.11.248 __short2tf32_dn	412
3.11.249 __short2tf32_oz	413
3.11.250 __short2tf32_rd	413
3.11.251 __short2tf32_rm	414
3.11.252 __short2tf32_rn	414
3.11.253 __short2tf32_tz	415
3.11.254 __short2tf32_up	415
3.11.255 __tf322bfloat16	416
3.11.256 __tf322bfloat16_dn	416
3.11.257 __tf322bfloat16_oz	417
3.11.258 __tf322bfloat16_rd	417
3.11.259 __tf322bfloat16_rm	418
3.11.260 __tf322bfloat16_rn	418
3.11.261 __tf322bfloat16_tz	419

3.11.262 __tf322bfloat16_up	419
3.11.263 __tf322char	420
3.11.264 __tf322char_dn	420
3.11.265 __tf322char_oz	421
3.11.266 __tf322char_rd	421
3.11.267 __tf322char_rm	422
3.11.268 __tf322char_rn	422
3.11.269 __tf322char_tz	423
3.11.270 __tf322char_up	423
3.11.271 __tf322float	424
3.11.272 __tf322float_dn	425
3.11.273 __tf322float_oz	425
3.11.274 __tf322float_rd	426
3.11.275 __tf322float_rm	426
3.11.276 __tf322float_rn	427
3.11.277 __tf322float_tz	427
3.11.278 __tf322float_up	428
3.11.279 __tf322half	428
3.11.280 __tf322half_dn	429
3.11.281 __tf322half_oz	429
3.11.282 __tf322half_rd	430
3.11.283 __tf322half_rm	430
3.11.284 __tf322half_rn	431
3.11.285 __tf322half_tz	431
3.11.286 __tf322half_up	432
3.11.287 __tf322int	432
3.11.288 __tf322int_dn	433
3.11.289 __tf322int_oz	433
3.11.290 __tf322int_rd	434
3.11.291 __tf322int_rm	434
3.11.292 __tf322int_rn	435
3.11.293 __tf322int_tz	435
3.11.294 __tf322int_up	436
3.11.295 __tf322short	436
3.11.296 __tf322short_dn	437
3.11.297 __tf322short_oz	438
3.11.298 __tf322short_rd	438
3.11.299 __tf322short_rm	439
3.11.300 __tf322short_rn	439
3.11.301 __tf322short_tz	440
3.11.302 __tf322short_up	440
3.11.303 __uchar2bfloat16	441
3.11.304 __uchar2float	441
3.11.305 __uchar2half	442
3.11.306 __uchar2tf32	442
3.11.307 __uint2bfloat16	443
3.11.308 __uint2bfloat16_dn	444

3.11.309 __uint2bfloat16_oz	444
3.11.310 __uint2bfloat16_rd	445
3.11.311 __uint2bfloat16_rm	445
3.11.312 __uint2bfloat16_rn	446
3.11.313 __uint2bfloat16_tz	446
3.11.314 __uint2bfloat16_up	447
3.11.315 __uint2float	447
3.11.316 __uint2float_dn	448
3.11.317 __uint2float_oz	448
3.11.318 __uint2float_rd	449
3.11.319 __uint2float_rm	449
3.11.320 __uint2float_rn	450
3.11.321 __uint2float_tz	450
3.11.322 __uint2float_up	451
3.11.323 __uint2half	451
3.11.324 __uint2half_dn	452
3.11.325 __uint2half_oz	452
3.11.326 __uint2half_rd	453
3.11.327 __uint2half_rm	453
3.11.328 __uint2half_rn	454
3.11.329 __uint2half_tz	454
3.11.330 __uint2half_up	455
3.11.331 __uint2tf32	455
3.11.332 __uint2tf32_dn	456
3.11.333 __uint2tf32_oz	457
3.11.334 __uint2tf32_rd	457
3.11.335 __uint2tf32_rm	458
3.11.336 __uint2tf32_rn	458
3.11.337 __uint2tf32_tz	459
3.11.338 __uint2tf32_up	459
3.11.339 __ushort2bfloat16	460
3.11.340 __ushort2bfloat16_dn	460
3.11.341 __ushort2bfloat16_oz	461
3.11.342 __ushort2bfloat16_rd	461
3.11.343 __ushort2bfloat16_rm	462
3.11.344 __ushort2bfloat16_rn	462
3.11.345 __ushort2bfloat16_tz	463
3.11.346 __ushort2bfloat16_up	463
3.11.347 __ushort2float	464
3.11.348 __ushort2half	464
3.11.349 __ushort2half_dn	465
3.11.350 __ushort2half_oz	465
3.11.351 __ushort2half_rd	466
3.11.352 __ushort2half_rm	466
3.11.353 __ushort2half_rn	467
3.11.354 __ushort2half_tz	467
3.11.355 __ushort2half_up	468

3.11.356 __ushort2tf32	468
3.11.357 __ushort2tf32_dn	469
3.11.358 __ushort2tf32_oz	470
3.11.359 __ushort2tf32_rd	470
3.11.360 __ushort2tf32_rm	471
3.11.361 __ushort2tf32_rn	471
3.11.362 __ushort2tf32_tz	472
3.11.363 __ushort2tf32_up	472
3.12 Synchronization Functions	473
3.12.1 __sync_all	473
3.12.2 __sync_all_ipu	473
3.12.3 __sync_all_mpu	474
3.12.4 __sync_cluster	474
3.12.5 __sync_compute	475
3.12.6 __sync_io	475
3.12.7 __sync_io_move_compute	476
3.12.8 __sync_move	476
3.13 Vector Active Functions	477
3.13.1 __bang_active_abs	477
3.13.2 __bang_active_cos	478
3.13.3 __bang_active_exp	478
3.13.4 __bang_active_exp_less_0	479
3.13.5 __bang_active_exphp	480
3.13.6 __bang_active_gelu	481
3.13.7 __bang_active_gelup	482
3.13.8 __bang_active_log	483
3.13.9 __bang_active_loghp	484
3.13.10 __bang_active_pow2	485
3.13.11 __bang_active_recip	486
3.13.12 __bang_active_recip_greater_1	487
3.13.13 __bang_active_reciphp	488
3.13.14 __bang_active_relu	489
3.13.15 __bang_active_rsqrt	489
3.13.16 __bang_active_rsqrthp	490
3.13.17 __bang_active_sigmoid	491
3.13.18 __bang_active_sign	492
3.13.19 __bang_active_sin	493
3.13.20 __bang_active_sqrt	493
3.13.21 __bang_active_sqrthp	494
3.13.22 __bang_active_tanh	495
3.13.23 __bang_taylor3_cos	496
3.13.24 __bang_taylor3_sigmoid	497
3.13.25 __bang_taylor3_sin	499
3.13.26 __bang_taylor3_softplus	501
3.13.27 __bang_taylor3_tanh	503
3.13.28 __bang_taylor4_cos	504
3.13.29 __bang_taylor4_sigmoid	505

3.13.30	__bang_taylor4_sin	507
3.13.31	__bang_taylor4_softplus	508
3.13.32	__bang_taylor4_tanh	510
3.14	Vector Bitwise Functions	511
3.14.1	__bang_band	511
3.14.2	__bang_band_scalar	512
3.14.3	__bang_bnot	513
3.14.4	__bang_bor	514
3.14.5	__bang_bor_scalar	514
3.14.6	__bang_bxor	516
3.14.7	__bang_bxor_scalar	516
3.14.8	__bang_cycle_band	518
3.14.9	__bang_cycle_bor	519
3.14.10	__bang_cycle_bxor	520
3.15	Vector Comparison Functions	520
3.15.1	__bang_argmax	520
3.15.2	__bang_argmin	522
3.15.3	__bang_cycle_eq	523
3.15.4	__bang_cycle_equ	525
3.15.5	__bang_cycle_ge	526
3.15.6	__bang_cycle_geu	528
3.15.7	__bang_cycle_gt	529
3.15.8	__bang_cycle_gtu	530
3.15.9	__bang_cycle_le	531
3.15.10	__bang_cycle_leu	533
3.15.11	__bang_cycle_lt	534
3.15.12	__bang_cycle_ltu	536
3.15.13	__bang_cycle_maxequal	537
3.15.14	__bang_cycle_maximum	539
3.15.15	__bang_cycle_minequal	540
3.15.16	__bang_cycle_minimum	542
3.15.17	__bang_cycle_nan_maximum	543
3.15.18	__bang_cycle_nan_minimum	544
3.15.19	__bang_cycle_ne	545
3.15.20	__bang_cycle_neu	547
3.15.21	__bang_eq	548
3.15.22	__bang_eq_bitindex	550
3.15.23	__bang_eq_scalar	551
3.15.24	__bang_equ	553
3.15.25	__bang_equ_bitindex	554
3.15.26	__bang_equ_scalar	555
3.15.27	__bang_ge	556
3.15.28	__bang_ge_bitindex	558
3.15.29	__bang_ge_scalar	560
3.15.30	__bang_geu	562
3.15.31	__bang_geu_bitindex	563
3.15.32	__bang_geu_scalar	564

3.15.33	<code>__bang_gt</code>	565
3.15.34	<code>__bang_gt_bitindex</code>	566
3.15.35	<code>__bang_gt_scalar</code>	567
3.15.36	<code>__bang_gtu</code>	568
3.15.37	<code>__bang_gtu_bitindex</code>	569
3.15.38	<code>__bang_gtu_scalar</code>	570
3.15.39	<code>__bang_le</code>	571
3.15.40	<code>__bang_le_bitindex</code>	572
3.15.41	<code>__bang_le_scalar</code>	574
3.15.42	<code>__bang_leu</code>	575
3.15.43	<code>__bang_leu_bitindex</code>	576
3.15.44	<code>__bang_leu_scalar</code>	577
3.15.45	<code>__bang_lt</code>	578
3.15.46	<code>__bang_lt_bitindex</code>	579
3.15.47	<code>__bang_lt_scalar</code>	581
3.15.48	<code>__bang_ltu</code>	583
3.15.49	<code>__bang_ltu_bitindex</code>	584
3.15.50	<code>__bang_ltu_scalar</code>	585
3.15.51	<code>__bang_max</code>	586
3.15.52	<code>__bang_maxeq_scalar</code>	587
3.15.53	<code>__bang_maxequal</code>	588
3.15.54	<code>__bang_maximum</code>	590
3.15.55	<code>__bang_min</code>	591
3.15.56	<code>__bang_mineq_scalar</code>	592
3.15.57	<code>__bang_minequal</code>	593
3.15.58	<code>__bang_minimum</code>	595
3.15.59	<code>__bang_nan_maximum</code>	596
3.15.60	<code>__bang_nan_minimum</code>	597
3.15.61	<code>__bang_ne</code>	598
3.15.62	<code>__bang_ne_bitindex</code>	599
3.15.63	<code>__bang_ne_scalar</code>	601
3.15.64	<code>__bang_neu</code>	602
3.15.65	<code>__bang_neu_bitindex</code>	603
3.15.66	<code>__bang_neu_scalar</code>	604
3.16	Vector Fusion Functions	605
3.16.1	<code>__bang fabsmax</code>	605
3.16.2	<code>__bang fabsmin</code>	606
3.16.3	<code>__bang fcmpfilter</code>	607
3.16.4	<code>__bang_fusion</code>	609
3.17	Vector Logic Functions	613
3.17.1	<code>__bang_and</code>	613
3.17.2	<code>__bang_and_scalar</code>	615
3.17.3	<code>__bang_cycle_and</code>	617
3.17.4	<code>__bang_cycle_or</code>	619
3.17.5	<code>__bang_cycle_xor</code>	620
3.17.6	<code>__bang_not</code>	622
3.17.7	<code>__bang_or</code>	623

3.17.8	<code>__bang_or_scalar</code>	624
3.17.9	<code>__bang_xor</code>	626
3.17.10	<code>__bang_xor_scalar</code>	627
3.18	Vector Movement Functions	629
3.18.1	<code>__bang_collect</code>	629
3.18.2	<code>__bang_collect_bitindex</code>	630
3.18.3	<code>__bang_maskmove</code>	631
3.18.4	<code>__bang_mirror</code>	632
3.18.5	<code>__bang_pad</code>	634
3.18.6	<code>__bang_rotate180</code>	639
3.18.7	<code>__bang_rotate270</code>	640
3.18.8	<code>__bang_rotate90</code>	642
3.18.9	<code>__bang_select</code>	644
3.18.10	<code>__bang_select_bitindex</code>	645
3.18.11	<code>__bang_tiling_2d_b128</code>	646
3.18.12	<code>__bang_tiling_2d_b16</code>	647
3.18.13	<code>__bang_tiling_2d_b256</code>	648
3.18.14	<code>__bang_tiling_2d_b32</code>	649
3.18.15	<code>__bang_tiling_2d_b64</code>	651
3.18.16	<code>__bang_tiling_2d_b8</code>	652
3.18.17	<code>__bang_tiling_3d_b1024</code>	653
3.18.18	<code>__bang_transpose</code>	655
3.19	Vector Operation Functions	657
3.19.1	<code>__bang_abs</code>	657
3.19.2	<code>__bang_add</code>	658
3.19.3	<code>__bang_add_scalar</code>	660
3.19.4	<code>__bang_add_tz</code>	661
3.19.5	<code>__bang_adds</code>	662
3.19.6	<code>__bang_adds_scalar</code>	664
3.19.7	<code>__bang_count</code>	665
3.19.8	<code>__bang_count_bitindex</code>	666
3.19.9	<code>__bang_cycle_add</code>	667
3.19.10	<code>__bang_cycle_add_tz</code>	669
3.19.11	<code>__bang_cycle_adds</code>	670
3.19.12	<code>__bang_cycle_mul</code>	672
3.19.13	<code>__bang_cycle_mulh</code>	673
3.19.14	<code>__bang_cycle_muls</code>	675
3.19.15	<code>__bang_cycle_sub</code>	676
3.19.16	<code>__bang_cycle_subs</code>	677
3.19.17	<code>__bang_findfirst1</code>	679
3.19.18	<code>__bang_findlast1</code>	680
3.19.19	<code>__bang_floor</code>	680
3.19.20	<code>__bang_histogram</code>	681
3.19.21	<code>__bang_integral</code>	685
3.19.22	<code>__bang_lut_s16</code>	687
3.19.23	<code>__bang_lut_s32</code>	688
3.19.24	<code>__bang_mul</code>	689

3.19.25	<code>__bang_mul_scalar</code>	690
3.19.26	<code>__bang_mulh</code>	692
3.19.27	<code>__bang_mulh_scalar</code>	693
3.19.28	<code>__bang_muls</code>	694
3.19.29	<code>__bang_muls_scalar</code>	695
3.19.30	<code>__bang_nearbyint</code>	696
3.19.31	<code>__bang_neg</code>	696
3.19.32	<code>__bang_nsa</code>	697
3.19.33	<code>__bang_popcnt</code>	698
3.19.34	<code>__bang_rand</code>	699
3.19.35	<code>__bang_reduce_sum</code>	700
3.19.36	<code>__bang_relu</code>	701
3.19.37	<code>__bang_relun</code>	702
3.19.38	<code>__bang_rol</code>	704
3.19.39	<code>__bang_ror</code>	705
3.19.40	<code>__bang_round</code>	706
3.19.41	<code>__bang_set0in32</code>	706
3.19.42	<code>__bang_set1in32</code>	707
3.19.43	<code>__bang_sll</code>	708
3.19.44	<code>__bang_square</code>	709
3.19.45	<code>__bang_sra</code>	710
3.19.46	<code>__bang_srl</code>	711
3.19.47	<code>__bang_sub</code>	712
3.19.48	<code>__bang_sub_scalar</code>	713
3.19.49	<code>__bang_subs</code>	715
3.19.50	<code>__bang_subs_scalar</code>	716
3.19.51	<code>__bang_sum</code>	717
3.20	Vector Surpass Functions	717
3.20.1	<code>__bang_cos</code>	717
3.20.2	<code>__bang_div</code>	718
3.20.3	<code>__bang_log</code>	720
3.20.4	<code>__bang_pow2</code>	721
3.20.5	<code>__bang_recip</code>	722
3.20.6	<code>__bang_rem</code>	722
3.20.7	<code>__bang_rsqrt</code>	724
3.20.8	<code>__bang_sin</code>	725
3.20.9	<code>__bang_sqrt</code>	726
3.21	Vector Type Conversion Functions	726
3.21.1	<code>__bang_bfloat162float</code>	726
3.21.2	<code>__bang_bfloat162half</code>	727
3.21.3	<code>__bang_bfloat162half_dn</code>	728
3.21.4	<code>__bang_bfloat162half_oz</code>	729
3.21.5	<code>__bang_bfloat162half_rd</code>	730
3.21.6	<code>__bang_bfloat162half_rm</code>	730
3.21.7	<code>__bang_bfloat162half_rn</code>	731
3.21.8	<code>__bang_bfloat162half_tz</code>	732
3.21.9	<code>__bang_bfloat162half_up</code>	733

3.21.10	<code>__bang_bfloat162int16</code>	734
3.21.11	<code>__bang_bfloat162int16_dn</code>	735
3.21.12	<code>__bang_bfloat162int16_oz</code>	736
3.21.13	<code>__bang_bfloat162int16_rd</code>	737
3.21.14	<code>__bang_bfloat162int16_rm</code>	738
3.21.15	<code>__bang_bfloat162int16_rn</code>	739
3.21.16	<code>__bang_bfloat162int16_tz</code>	740
3.21.17	<code>__bang_bfloat162int16_up</code>	741
3.21.18	<code>__bang_bfloat162int32</code>	742
3.21.19	<code>__bang_bfloat162int32_dn</code>	743
3.21.20	<code>__bang_bfloat162int32_oz</code>	744
3.21.21	<code>__bang_bfloat162int32_rd</code>	745
3.21.22	<code>__bang_bfloat162int32_rm</code>	746
3.21.23	<code>__bang_bfloat162int32_rn</code>	747
3.21.24	<code>__bang_bfloat162int32_tz</code>	748
3.21.25	<code>__bang_bfloat162int32_up</code>	749
3.21.26	<code>__bang_bfloat162int4_dn</code>	750
3.21.27	<code>__bang_bfloat162int4_oz</code>	751
3.21.28	<code>__bang_bfloat162int4_rd</code>	752
3.21.29	<code>__bang_bfloat162int4_rm</code>	753
3.21.30	<code>__bang_bfloat162int4_rn</code>	754
3.21.31	<code>__bang_bfloat162int4_tz</code>	755
3.21.32	<code>__bang_bfloat162int4_up</code>	756
3.21.33	<code>__bang_bfloat162int8</code>	757
3.21.34	<code>__bang_bfloat162int8_dn</code>	758
3.21.35	<code>__bang_bfloat162int8_oz</code>	759
3.21.36	<code>__bang_bfloat162int8_rd</code>	760
3.21.37	<code>__bang_bfloat162int8_rm</code>	761
3.21.38	<code>__bang_bfloat162int8_rn</code>	762
3.21.39	<code>__bang_bfloat162int8_tz</code>	763
3.21.40	<code>__bang_bfloat162int8_up</code>	764
3.21.41	<code>__bang_bfloat162tf32</code>	765
3.21.42	<code>__bang_bfloat162uchar</code>	765
3.21.43	<code>__bang_bfloat162uchar_dn</code>	766
3.21.44	<code>__bang_bfloat162uchar_oz</code>	767
3.21.45	<code>__bang_bfloat162uchar_rd</code>	768
3.21.46	<code>__bang_bfloat162uchar_rm</code>	769
3.21.47	<code>__bang_bfloat162uchar_rn</code>	769
3.21.48	<code>__bang_bfloat162uchar_tz</code>	770
3.21.49	<code>__bang_bfloat162uchar_up</code>	771
3.21.50	<code>__bang_float2bfloat16_dn</code>	772
3.21.51	<code>__bang_float2bfloat16_oz</code>	772
3.21.52	<code>__bang_float2bfloat16_rd</code>	773
3.21.53	<code>__bang_float2bfloat16_rm</code>	774
3.21.54	<code>__bang_float2bfloat16_rn</code>	775
3.21.55	<code>__bang_float2bfloat16_tz</code>	775
3.21.56	<code>__bang_float2bfloat16_up</code>	776

3.21.57 __bang_float2half_dn	777
3.21.58 __bang_float2half_oz	778
3.21.59 __bang_float2half_rd	779
3.21.60 __bang_float2half_rm	780
3.21.61 __bang_float2half_rn	781
3.21.62 __bang_float2half_sr	782
3.21.63 __bang_float2half_tz	782
3.21.64 __bang_float2half_up	784
3.21.65 __bang_float2int16_dn	785
3.21.66 __bang_float2int16_oz	786
3.21.67 __bang_float2int16_rd	787
3.21.68 __bang_float2int16_rm	787
3.21.69 __bang_float2int16_rn	788
3.21.70 __bang_float2int16_tz	789
3.21.71 __bang_float2int16_up	790
3.21.72 __bang_float2int32	790
3.21.73 __bang_float2int32_dn	791
3.21.74 __bang_float2int32_oz	792
3.21.75 __bang_float2int32_rd	793
3.21.76 __bang_float2int32_rm	794
3.21.77 __bang_float2int32_rn	795
3.21.78 __bang_float2int32_tz	796
3.21.79 __bang_float2int32_up	797
3.21.80 __bang_float2int4_dn	798
3.21.81 __bang_float2int4_oz	799
3.21.82 __bang_float2int4_rd	800
3.21.83 __bang_float2int4_rm	801
3.21.84 __bang_float2int4_rn	802
3.21.85 __bang_float2int4_tz	803
3.21.86 __bang_float2int4_up	804
3.21.87 __bang_float2int8_dn	805
3.21.88 __bang_float2int8_oz	806
3.21.89 __bang_float2int8_rd	806
3.21.90 __bang_float2int8_rm	807
3.21.91 __bang_float2int8_rn	808
3.21.92 __bang_float2int8_tz	809
3.21.93 __bang_float2int8_up	809
3.21.94 __bang_float2tf32	810
3.21.95 __bang_float2tf32_dn	811
3.21.96 __bang_float2tf32_oz	812
3.21.97 __bang_float2tf32_rd	813
3.21.98 __bang_float2tf32_rm	813
3.21.99 __bang_float2tf32_rn	814
3.21.100 __bang_float2tf32_sr	815
3.21.101 __bang_float2tf32_tz	816
3.21.102 __bang_float2tf32_up	816
3.21.103 __bang_float2uchar	817

3.21.104 __bang_float2uchar_dn	818
3.21.105 __bang_float2uchar_oz	819
3.21.106 __bang_float2uchar_rd	820
3.21.107 __bang_float2uchar_rm	821
3.21.108 __bang_float2uchar_rn	822
3.21.109 __bang_float2uchar_tz	823
3.21.110 __bang_float2uchar_up	824
3.21.111 __bang_half2bfloat16	825
3.21.112 __bang_half2bfloat16_dn	825
3.21.113 __bang_half2bfloat16_oz	826
3.21.114 __bang_half2bfloat16_rd	827
3.21.115 __bang_half2bfloat16_rm	828
3.21.116 __bang_half2bfloat16_rn	829
3.21.117 __bang_half2bfloat16_tz	829
3.21.118 __bang_half2bfloat16_up	830
3.21.119 __bang_half2float	831
3.21.120 __bang_half2int16_dn	832
3.21.121 __bang_half2int16_oz	833
3.21.122 __bang_half2int16_rd	834
3.21.123 __bang_half2int16_rm	835
3.21.124 __bang_half2int16_rm	836
3.21.125 __bang_half2int16_tz	836
3.21.126 __bang_half2int16_up	837
3.21.127 __bang_half2int32	838
3.21.128 __bang_half2int32_dn	839
3.21.129 __bang_half2int32_oz	840
3.21.130 __bang_half2int32_rd	841
3.21.131 __bang_half2int32_rm	842
3.21.132 __bang_half2int32_rn	843
3.21.133 __bang_half2int32_tz	844
3.21.134 __bang_half2int32_up	845
3.21.135 __bang_half2int4_dn	846
3.21.136 __bang_half2int4_oz	847
3.21.137 __bang_half2int4_rd	848
3.21.138 __bang_half2int4_rm	849
3.21.139 __bang_half2int4_rn	850
3.21.140 __bang_half2int4_tz	851
3.21.141 __bang_half2int4_up	852
3.21.142 __bang_half2int8_dn	853
3.21.143 __bang_half2int8_oz	855
3.21.144 __bang_half2int8_rd	856
3.21.145 __bang_half2int8_rm	857
3.21.146 __bang_half2int8_rn	858
3.21.147 __bang_half2int8_tz	859
3.21.148 __bang_half2int8_up	860
3.21.149 __bang_half2short_dn	861
3.21.150 __bang_half2short_oz	863

3.21.151 __bang_half2short_rd	864
3.21.152 __bang_half2short_rm	865
3.21.153 __bang_half2short_rn	866
3.21.154 __bang_half2short_tz	866
3.21.155 __bang_half2short_up	868
3.21.156 __bang_half2tf32	869
3.21.157 __bang_half2uchar_dn	870
3.21.158 __bang_int162bfloat16	871
3.21.159 __bang_int162bfloat16_dn	872
3.21.160 __bang_int162bfloat16_oz	873
3.21.161 __bang_int162bfloat16_rd	874
3.21.162 __bang_int162bfloat16_rm	875
3.21.163 __bang_int162bfloat16_rn	876
3.21.164 __bang_int162bfloat16_tz	877
3.21.165 __bang_int162bfloat16_up	878
3.21.166 __bang_int162float	879
3.21.167 __bang_int162half	880
3.21.168 __bang_int162int32	881
3.21.169 __bang_int162int4_dn	882
3.21.170 __bang_int162int4_oz	883
3.21.171 __bang_int162int4_rd	884
3.21.172 __bang_int162int4_rm	885
3.21.173 __bang_int162int4_rn	886
3.21.174 __bang_int162int4_tz	887
3.21.175 __bang_int162int4_up	888
3.21.176 __bang_int162int8	889
3.21.177 __bang_int162tf32	890
3.21.178 __bang_int162tf32_dn	891
3.21.179 __bang_int162tf32_oz	892
3.21.180 __bang_int162tf32_rd	893
3.21.181 __bang_int162tf32_rm	894
3.21.182 __bang_int162tf32_rn	895
3.21.183 __bang_int162tf32_tz	896
3.21.184 __bang_int162tf32_up	897
3.21.185 __bang_int162uchar	898
3.21.186 __bang_int162uchar_dn	899
3.21.187 __bang_int162uchar_oz	900
3.21.188 __bang_int162uchar_rd	901
3.21.189 __bang_int162uchar_rm	902
3.21.190 __bang_int162uchar_rn	903
3.21.191 __bang_int162uchar_tz	904
3.21.192 __bang_int162uchar_up	905
3.21.193 __bang_int322bfloat16	906
3.21.194 __bang_int322bfloat16_dn	907
3.21.195 __bang_int322bfloat16_oz	908
3.21.196 __bang_int322bfloat16_rd	909
3.21.197 __bang_int322bfloat16_rm	910

3.21.198 __bang_int322bfloat16_rn	911
3.21.199 __bang_int322bfloat16_tz	912
3.21.200 __bang_int322bfloat16_up	913
3.21.201 __bang_int322float	914
3.21.202 __bang_int322float_dn	915
3.21.203 __bang_int322float_oz	916
3.21.204 __bang_int322float_rd	917
3.21.205 __bang_int322float_rm	918
3.21.206 __bang_int322float_rn	919
3.21.207 __bang_int322float_tz	920
3.21.208 __bang_int322float_up	921
3.21.209 __bang_int322half	922
3.21.210 __bang_int322half_dn	923
3.21.211 __bang_int322half_oz	924
3.21.212 __bang_int322half_rd	925
3.21.213 __bang_int322half_rm	926
3.21.214 __bang_int322half_rn	927
3.21.215 __bang_int322half_tz	928
3.21.216 __bang_int322half_up	929
3.21.217 __bang_int322int16	930
3.21.218 __bang_int322int4_dn	931
3.21.219 __bang_int322int4_oz	932
3.21.220 __bang_int322int4_rd	933
3.21.221 __bang_int322int4_rm	934
3.21.222 __bang_int322int4_rn	935
3.21.223 __bang_int322int4_tz	936
3.21.224 __bang_int322int4_up	937
3.21.225 __bang_int322int8	938
3.21.226 __bang_int322tf32	939
3.21.227 __bang_int322tf32_dn	940
3.21.228 __bang_int322tf32_oz	941
3.21.229 __bang_int322tf32_rd	942
3.21.230 __bang_int322tf32_rm	943
3.21.231 __bang_int322tf32_rn	944
3.21.232 __bang_int322tf32_tz	945
3.21.233 __bang_int322tf32_up	946
3.21.234 __bang_int322uchar	947
3.21.235 __bang_int322uchar_dn	948
3.21.236 __bang_int322uchar_oz	949
3.21.237 __bang_int322uchar_rd	950
3.21.238 __bang_int322uchar_rm	951
3.21.239 __bang_int322uchar_rn	952
3.21.240 __bang_int322uchar_tz	953
3.21.241 __bang_int322uchar_up	954
3.21.242 __bang_int42bfloat16_dn	955
3.21.243 __bang_int42bfloat16_oz	956
3.21.244 __bang_int42bfloat16_rd	957

3.21.245 __bang_int42bfloat16_rm	958
3.21.246 __bang_int42bfloat16_rn	959
3.21.247 __bang_int42bfloat16_tz	960
3.21.248 __bang_int42bfloat16_up	961
3.21.249 __bang_int42bfloat_dn	962
3.21.250 __bang_int42float_oz	963
3.21.251 __bang_int42float_rd	964
3.21.252 __bang_int42float_rm	965
3.21.253 __bang_int42float_rn	966
3.21.254 __bang_int42float_tz	967
3.21.255 __bang_int42float_up	968
3.21.256 __bang_int42half_dn	969
3.21.257 __bang_int42half_oz	970
3.21.258 __bang_int42half_rd	971
3.21.259 __bang_int42half_rm	972
3.21.260 __bang_int42half_rn	973
3.21.261 __bang_int42half_tz	974
3.21.262 __bang_int42half_up	975
3.21.263 __bang_int42int16_dn	976
3.21.264 __bang_int42int16_oz	977
3.21.265 __bang_int42int16_rd	978
3.21.266 __bang_int42int16_rm	979
3.21.267 __bang_int42int16_rn	980
3.21.268 __bang_int42int16_tz	981
3.21.269 __bang_int42int16_up	982
3.21.270 __bang_int42int32_dn	983
3.21.271 __bang_int42int32_oz	984
3.21.272 __bang_int42int32_rd	985
3.21.273 __bang_int42int32_rm	986
3.21.274 __bang_int42int32_rn	987
3.21.275 __bang_int42int32_tz	988
3.21.276 __bang_int42int32_up	989
3.21.277 __bang_int42int8_dn	990
3.21.278 __bang_int42int8_oz	991
3.21.279 __bang_int42int8_rd	992
3.21.280 __bang_int42int8_rm	993
3.21.281 __bang_int42int8_rn	994
3.21.282 __bang_int42int8_tz	995
3.21.283 __bang_int42int8_up	996
3.21.284 __bang_int42tf32	997
3.21.285 __bang_int42tf32_dn	998
3.21.286 __bang_int42tf32_oz	999
3.21.287 __bang_int42tf32_rd	1000
3.21.288 __bang_int42tf32_rm	1001
3.21.289 __bang_int42tf32_rn	1002
3.21.290 __bang_int42tf32_tz	1003
3.21.291 __bang_int42tf32_up	1004

3.21.292 __bang_int42uchar	1005
3.21.293 __bang_int42uchar_dn	1006
3.21.294 __bang_int42uchar_oz	1007
3.21.295 __bang_int42uchar_rd	1008
3.21.296 __bang_int42uchar_rm	1009
3.21.297 __bang_int42uchar_rn	1010
3.21.298 __bang_int42uchar_tz	1011
3.21.299 __bang_int42uchar_up	1012
3.21.300 __bang_int82bfloat16	1013
3.21.301 __bang_int82bfloat16_dn	1014
3.21.302 __bang_int82bfloat16_oz	1015
3.21.303 __bang_int82bfloat16_rd	1016
3.21.304 __bang_int82bfloat16_rm	1017
3.21.305 __bang_int82bfloat16_rn	1018
3.21.306 __bang_int82bfloat16_tz	1019
3.21.307 __bang_int82bfloat16_up	1020
3.21.308 __bang_int82float	1021
3.21.309 __bang_int82half	1022
3.21.310 __bang_int82int16	1023
3.21.311 __bang_int82int32	1024
3.21.312 __bang_int82int4_dn	1025
3.21.313 __bang_int82int4_oz	1026
3.21.314 __bang_int82int4_rd	1027
3.21.315 __bang_int82int4_rm	1028
3.21.316 __bang_int82int4_rn	1029
3.21.317 __bang_int82int4_tz	1030
3.21.318 __bang_int82int4_up	1031
3.21.319 __bang_int82tf32	1032
3.21.320 __bang_int82tf32_dn	1033
3.21.321 __bang_int82tf32_oz	1034
3.21.322 __bang_int82tf32_rd	1035
3.21.323 __bang_int82tf32_rm	1036
3.21.324 __bang_int82tf32_rn	1037
3.21.325 __bang_int82tf32_tz	1038
3.21.326 __bang_int82tf32_up	1039
3.21.327 __bang_short2half	1040
3.21.328 __bang_tf322bfloat16	1041
3.21.329 __bang_tf322bfloat16_dn	1042
3.21.330 __bang_tf322bfloat16_oz	1043
3.21.331 __bang_tf322bfloat16_rd	1044
3.21.332 __bang_tf322bfloat16_rm	1044
3.21.333 __bang_tf322bfloat16_rn	1045
3.21.334 __bang_tf322bfloat16_tz	1046
3.21.335 __bang_tf322bfloat16_up	1047
3.21.336 __bang_tf322float	1048
3.21.337 __bang_tf322float_dn	1048
3.21.338 __bang_tf322float_oz	1049

3.21.339 __bang_tf32float_rd	1050
3.21.340 __bang_tf32float_rm	1051
3.21.341 __bang_tf32float_rn	1052
3.21.342 __bang_tf32float_tz	1052
3.21.343 __bang_tf32float_up	1053
3.21.344 __bang_tf32half	1054
3.21.345 __bang_tf32half_dn	1055
3.21.346 __bang_tf32half_oz	1056
3.21.347 __bang_tf32half_rd	1056
3.21.348 __bang_tf32half_rm	1057
3.21.349 __bang_tf32half_rn	1058
3.21.350 __bang_tf32half_tz	1059
3.21.351 __bang_tf32half_up	1060
3.21.352 __bang_tf32int16	1060
3.21.353 __bang_tf32int16_dn	1061
3.21.354 __bang_tf32int16_oz	1062
3.21.355 __bang_tf32int16_rd	1063
3.21.356 __bang_tf32int16_rm	1064
3.21.357 __bang_tf32int16_rn	1065
3.21.358 __bang_tf32int16_tz	1066
3.21.359 __bang_tf32int16_up	1067
3.21.360 __bang_tf32int32	1068
3.21.361 __bang_tf32int32_dn	1069
3.21.362 __bang_tf32int32_oz	1070
3.21.363 __bang_tf32int32_rd	1071
3.21.364 __bang_tf32int32_rm	1072
3.21.365 __bang_tf32int32_rn	1073
3.21.366 __bang_tf32int32_tz	1074
3.21.367 __bang_tf32int32_up	1075
3.21.368 __bang_tf32int4	1076
3.21.369 __bang_tf32int4_dn	1077
3.21.370 __bang_tf32int4_oz	1078
3.21.371 __bang_tf32int4_rd	1079
3.21.372 __bang_tf32int4_rm	1080
3.21.373 __bang_tf32int4_rn	1081
3.21.374 __bang_tf32int4_tz	1082
3.21.375 __bang_tf32int4_up	1083
3.21.376 __bang_tf32int8	1084
3.21.377 __bang_tf32int8_dn	1085
3.21.378 __bang_tf32int8_oz	1086
3.21.379 __bang_tf32int8_rd	1087
3.21.380 __bang_tf32int8_rm	1088
3.21.381 __bang_tf32int8_rn	1089
3.21.382 __bang_tf32int8_tz	1090
3.21.383 __bang_tf32int8_up	1091
3.21.384 __bang_tf32uchar	1092
3.21.385 __bang_tf32uchar_dn	1092

3.21.386 __bang_tf322uchar_oz	1093
3.21.387 __bang_tf322uchar_rd	1094
3.21.388 __bang_tf322uchar_rm	1095
3.21.389 __bang_tf322uchar_rn	1096
3.21.390 __bang_tf322uchar_tz	1096
3.21.391 __bang_tf322uchar_up	1097
3.21.392 __bang_uchar2bfloat16	1098
3.21.393 __bang_uchar2float	1099
3.21.394 __bang_uchar2half	1100
3.21.395 __bang_uchar2int16	1101
3.21.396 __bang_uchar2int16_dn	1102
3.21.397 __bang_uchar2int16_oz	1103
3.21.398 __bang_uchar2int16_rd	1104
3.21.399 __bang_uchar2int16_rm	1105
3.21.400 __bang_uchar2int16_rn	1106
3.21.401 __bang_uchar2int16_tz	1107
3.21.402 __bang_uchar2int16_up	1108
3.21.403 __bang_uchar2int32	1109
3.21.404 __bang_uchar2int32_dn	1110
3.21.405 __bang_uchar2int32_oz	1111
3.21.406 __bang_uchar2int32_rd	1112
3.21.407 __bang_uchar2int32_rm	1113
3.21.408 __bang_uchar2int32_rn	1114
3.21.409 __bang_uchar2int32_tz	1115
3.21.410 __bang_uchar2int32_up	1116
3.21.411 __bang_uchar2int4	1117
3.21.412 __bang_uchar2int4_dn	1118
3.21.413 __bang_uchar2int4_oz	1119
3.21.414 __bang_uchar2int4_rd	1120
3.21.415 __bang_uchar2int4_rm	1121
3.21.416 __bang_uchar2int4_rn	1122
3.21.417 __bang_uchar2int4_tz	1123
3.21.418 __bang_uchar2int4_up	1124
3.21.419 __bang_uchar2tf32	1125

4 Deprecated Built-in Functions	1126
4.1 __bang_add_const	1126
4.2 __bang_band	1127
4.3 __bang_bnot	1128
4.4 __bang_bor	1129
4.5 __bang_bxor	1130
4.6 __bang_char2int	1131
4.7 __bang_char2short	1132
4.8 __bang_count	1133
4.9 __bang_count_bitindex	1134
4.10 __bang_cycle_band	1135
4.11 __bang_cycle_bor	1136

4.12	<code>__bang_cycle_bxor</code>	1137
4.13	<code>__bang_div</code>	1138
4.14	<code>__bang_findfirst1</code>	1139
4.15	<code>__bang_findlast1</code>	1140
4.16	<code>__bang_fix82half</code>	1141
4.17	<code>__bang_float2int_dn</code>	1142
4.18	<code>__bang_float2int_oz</code>	1143
4.19	<code>__bang_float2int_rd</code>	1144
4.20	<code>__bang_float2int_rm</code>	1145
4.21	<code>__bang_float2int_rn</code>	1146
4.22	<code>__bang_float2int_tz</code>	1146
4.23	<code>__bang_float2int_up</code>	1147
4.24	<code>__bang_ge_const</code>	1148
4.25	<code>__bang_half2char_dn</code>	1149
4.26	<code>__bang_half2uchar_dn</code>	1150
4.27	<code>__bang_int2char</code>	1151
4.28	<code>__bang_int2float</code>	1152
4.29	<code>__bang_int2short</code>	1153
4.30	<code>__bang_lock</code>	1154
4.31	<code>__bang_maskmove</code>	1155
4.32	<code>__bang_maskmove_bitindex</code>	1156
4.33	<code>__bang_maximum</code>	1157
4.34	<code>__bang_mul_const</code>	1157
4.35	<code>__bang_pad</code>	1158
4.36	<code>__bang_short2char</code>	1162
4.37	<code>__bang_short2int</code>	1163
4.38	<code>__bang_sub_const</code>	1164
4.39	<code>__bang_unlock</code>	1165
4.40	<code>__memcpy_nram_to_nram</code>	1166
4.41	<code>__nramset</code>	1167
4.42	<code>__nramset_float</code>	1168
4.43	<code>__nramset_half</code>	1168
4.44	<code>__nramset_int</code>	1169
4.45	<code>__nramset_short</code>	1170
4.46	<code>__nramset_unsigned_int</code>	1170
4.47	<code>__nramset_unsigned_short</code>	1171

5	Appendix	1172
5.1	<code>Rounding Mode</code>	1172
5.2	<code>Floating Point Calculation</code>	1173
5.2.1	<code>Stream and Scalar Binary Operation Functions</code>	1173
5.2.2	<code>Stream and Scalar Unary Operation Functions</code>	1174
5.2.3	<code>Stream and Scalar Comparison Functions</code>	1174
5.2.4	<code>Element-wise Stream and Scalar Comparison Operation Functions</code>	1175
5.2.5	<code>Non-element-wise Stream Comparison Operation Functions</code>	1176
5.2.6	<code>Stream and Scalar Binary Logic and Bit Operation Functions</code>	1177
5.2.7	<code>Stream and Scalar Unary Logic and Bit Operation Functions</code>	1177

5.2.8	Stream and Scalar Type Conversion from Floating Point to Integer	1178
5.2.9	Stream and Scalar Type Conversion from Floating Point to Floating Point . .	1179
5.2.10	Stream and Scalar Type Conversion from Integer to Floating Point	1179
5.2.11	Stream and Scalar Div Operation Function	1180
5.3	Mathematical Functions	1180

Cambricon@155chb



List of Figures

3.1	Memory Copy Function with Stride	31
3.2	Asynchronous Memory Copy Function with Stride	37
3.3	3D Memory Copy Function with Stride	46
3.4	Process of Unpool Operation	198
3.5	Reciprocal Process	486
3.6	The Calculation Process of Half Type __bang_eq_bitindex	550
3.7	The Calculation Process of Half Type __bang_ge_bitindex	559
3.8	The Calculation Process of Half Type __bang_gt_bitindex	566
3.9	The Calculation Process of Half Type __bang_le_bitindex	573
3.10	The Calculation Process of Half Type __bang_lt_bitindex	580
3.11	The Calculation Process of Half Type __bang_ne_bitindex	600
3.12	The Calculation Process of float Type __bang_collect_bitindex	630
3.13	The Process of 2 Dimensional Tiling Function	652
3.14	The Process of 3 Dimensional Tiling Function	654
3.15	Description of __bang_cycle_add Operation	668
3.16	The Process of Conversion With Stride	853



List of Tables

2.1	Version Matrix	7
3.1	Alignment Constraints of Address Space in <code>__memcpy</code>	17
3.2	Direction Constraints of <code>__memcpy</code>	17
3.3	Alignment Constraints of <code>size</code> in <code>__memcpy</code>	18
3.4	Direction Constraints of <code>__memcpy_async</code>	20
3.5	Alignment Constraints of <code>size</code> in <code>__memcpy_async</code>	22
3.6	Alignment Constraints of Address Space in <code>__memcpy</code>	32
3.7	Direction Constraints of <code>__memcpy</code>	32
3.8	Alignment Constraints of <code>size</code> in <code>__memcpy</code>	34
3.9	Alignment Constraints of <code>dst_stride</code> in <code>__memcpy</code>	35
3.10	Alignment Constraints of <code>src_stride</code> in <code>__memcpy</code>	36
3.11	Direction Constraints of <code>__memcpy_async</code>	38
3.12	Alignment Constraints of <code>size</code> in <code>__memcpy_async</code>	39
3.13	Alignment Constraints of <code>dst_stride</code> in <code>__memcpy_async</code>	40
3.14	Alignment Constraints of <code>src_stride</code> in <code>__memcpy_async</code>	42
3.15	Semantics of <code>mluPoolBPOverlap</code>	64
3.16	Conv Data Types Supported on <code>(m)tp_2xx</code>	109
3.17	Conv Data Types Supported on <code>(m)tp_3xx</code>	110
3.18	Conv Partial Data Types Supported on <code>(m)tp_2xx</code>	137
3.19	Conv Partial Data Types Supported on <code>(m)tp_3xx</code>	138
3.20	Semantics of <code>mluPoolBPOverlap</code>	156
3.21	Data Types Supported on <code>(m)tp_2xx</code>	177
3.22	mlp Data Types Supported on <code>(m)tp_3xx</code>	177
3.23	Special Value of <code>__bang_taylor3_cos(half)</code>	497
3.24	Special Value of <code>__bang_taylor3_cos(float)</code>	497
3.25	Special Value of <code>__bang_taylor3_sigmoid(half)</code>	498
3.26	Special Value of <code>__bang_taylor3_sigmoid(float)</code>	499
3.27	Special Value of <code>__bang_taylor3_sin(half)</code>	500
3.28	Special Value of <code>__bang_taylor3_sin(float)</code>	500
3.29	Special Value of <code>__bang_taylor3_softplus(half)</code>	502
3.30	Special Value of <code>__bang_taylor3_softplus(float)</code>	502
3.31	Special Value of <code>__bang_taylor4_cos(half)</code>	504
3.32	Special Value of <code>__bang_taylor4_cos(float)</code>	505
3.33	Special Value of <code>__bang_taylor4_sigmoid(half)</code>	506
3.34	Special Value of <code>__bang_taylor4_sigmoid(float)</code>	506
3.35	Special Value of <code>__bang_taylor4_sin(half)</code>	507
3.36	Special Value of <code>__bang_taylor4_sin(float)</code>	508

3.37	Special Value of __bang_taylor4_softplus(half)	509
3.38	Special Value of __bang_taylor4_softplus(float)	509
3.39	Semantics of CompareMode	608
3.40	Semantics of mluFusionOpCode	612
3.41	Histogram Data Types Supported on (m)tp_220	683
3.42	Histogram Data Types Supported on mtp_372	683
3.43	Histogram Data Types Supported on tp_322	684
3.44	Histogram Data Types Supported on (m)tp_5xx	684
5.1	Rounding Mode	1172
5.2	Floating Point Calculation of Stream and Scalar Binary Operation Functions	1173
5.3	Floating Point Calculation of Stream and Scalar Unary Operation Functions	1174
5.4	Floating Point Calculation of Stream and Scalar Comparison Functions	1174
5.5	Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions	1175
5.6	Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions	1176
5.7	Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions	1177
5.8	Floating Point Calculation of Stream and Scalar Unary Logic and Bit Operation Functions	1177
5.9	Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer	1178
5.10	Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point	1179
5.11	Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point	1179
5.12	Floating Point Calculation of Stream and Scalar Div Operation Function	1180
5.13	Mathematical Functions and Macros	1180



1 Copyright

DISCLAIMER

CAMBRICON MAKES NO REPRESENTATION, WARRANTY (EXPRESS, IMPLIED, OR STATUTORY) OR GUARANTEE REGARDING THE INFORMATION CONTAINED HEREIN, AND EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, TITLE, NONINFRINGEMENT OF INTELLECTUAL PROPERTY OR FITNESS FOR A PARTICULAR PURPOSE, AND CAMBRICON DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR SERVICES. CAMBRICON SHALL HAVE NO LIABILITY RELATED TO ANY DEFAULTS, DAMAGES, COSTS OR PROBLEMS WHICH MAY BE BASED ON OR ATTRIBUTABLE TO: (I) THE USE OF THE CAMBRICON PRODUCT IN ANY MANNER THAT IS CONTRARY TO THIS GUIDE, OR (II) CUSTOMER PRODUCT DESIGNS.

LIMITATION OF LIABILITY

In no event shall Cambricon be liable for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption and loss of information) arising out of the use of or inability to use this guide, even if Cambricon has been advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever, Cambricon's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the Cambricon terms and conditions of sale for the product.

ACCURACY OF INFORMATION

Information provided in this document is proprietary to Cambricon, and Cambricon reserves the right to make any changes to the information in this document or to any products and services at any time without notice. The information contained in this guide and all other information contained in Cambricon documentation referenced in this guide is provided "AS IS." Cambricon does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within this guide. Cambricon may make changes to this guide, or to the products described therein, at any time without notice, but makes no commitment to update this guide.

Performance tests and ratings set forth in this guide are measured using specific chips or computer systems or components. The results shown in this guide reflect approximate performance of Cambricon products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. As set forth above, Cambricon makes no representation, warranty or guarantee that the product described in this guide will be suitable for any specified use. Cambricon does not represent or warrant that it tests all parameters of each product. It is customer's sole responsibility to ensure that the product is suitable and fit for the application planned by the customer and to do the necessary testing for the application in order to avoid a default of the application or the product.

Weaknesses in customer's product designs may affect the quality and reliability of Cambricon

1. COPYRIGHT

product and may result in additional or different conditions and/or requirements beyond those contained in this guide.

IP NOTICES

Cambricon and the Cambricon logo are trademarks and/or registered trademarks of Cambricon Corporation in China and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

This guide is copyrighted and is protected by worldwide copyright laws and treaty provisions. This guide may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without Cambricon's prior written permission. Other than the right for customer to use the information in this guide with the product, no other right or license, either express or implied, is granted by Cambricon under this guide. For the avoidance of doubt, Cambricon does not grant any right or license (express or implied) to customer under any patents, copyrights, trademarks, trade secret or any other intellectual property or proprietary rights of Cambricon.

- Copyright
- © 2022 Cambricon Corporation. All rights reserved.

Cambricon@155chb



2 Preface

2.1 Version

The following matrix indicates the supported architecture for different versions of Cambricon BANG C language, compiler, MLISA language, and assembler.

Cambricon@155chb

Table 2.1: Version Matrix

Cambricon BANG C Version	CNCC Version	MLISA Version	CNAS Version	Supported Arch
Cambricon BANG C-v4.0.x	CNCC-v4.0.x	MLISA-v4.0.x	CNAS-v4.0.x	mtp_592, mtp_372, tp_322, mtp_290, (m)tp_270, (m)tp_220, mtp_100, 1H8, 1H8 mini
Cambricon BANG C-v3.3.x	CNCC-v3.3.x	MLISA-v3.3.x	CNAS-v3.3.x	mtp_372, tp_322, mtp_290, (m)tp_270, (m)tp_220, mtp_100, 1H8, 1H8 mini
Cambricon BANG C-v3.0.x	CNCC-v3.0.x	MLISA-v3.0.x	CNAS-v3.0.x	mtp_372, mtp_290, (m)tp_270, (m)tp_220, mtp_100, 1H8, 1H8 mini
Cambricon BANG C-v2.16.x	CNCC-v2.16.x	MLISA-v2.16.x	CNAS-v2.16.x	mtp_290, (m)tp_270, (m)tp_220, mtp_100, 1H8, 1H8 mini
Cambricon BANG C-v2.0.x	CNCC-v2.0.x	MLISA-v2.0.x	CNAS-v2.0.x	mtp_270, mtp_220, mtp_100, 1H8, 1H8 mini
Cambricon BANG C-v1.7.x	CNCC-v1.7.x	MLISA-v1.8.x	CNAS-v1.8.x	mtp_100, 1H8, 1H8 mini
Cambricon BANG C-v1.3.x	CNCC-v1.3.x	MLISA-v1.4.x	CNAS-v1.4.x	mtp_100, 1H8
Cambricon BANG C-v1.0.x	CNCC-v1.0.x	MLISA-v1.1.x	CNAS-v1.1.x	mtp_100

Note:

The Cambricon BANG C v4.X is compatible with Cambricon BANG C v3.x.

2.2 Update History

- **V4.0.0**

Date : July 8, 2022

Changes :

- Add built-in functions including `__tf32float`, `__tf32float_tz`, `__tf32float_oz`, `__tf32float_up`, `__tf32float_dn`, `__tf32float_rd`, `__tf32float_rm`, `__tf32float_rn`, `__tf32half`, `__tf32half_tz`, `__tf32half_oz`, `__tf32half_up`, `__tf32half_dn`, `__tf32half_rd`, `__tf32half_rm`, `__tf32half_rn`, `__tf32short`, `__tf32short_tz`, `__tf32short_oz`, `__tf32short_up`, `__tf32short_dn`, `__tf32short_rd`, `__tf32short_rm`, `__tf32short_rn`, `__tf32int`, `__tf32int_tz`, `__tf32int_oz`, `__tf32int_up`, `__tf32int_dn`, `__tf32int_rd`, `__tf32int_rm`, `__tf32int_rn`, `__tf32char`, `__tf32char_tz`, `__tf32char_oz`, `__tf32char_up`, `__tf32char_dn`, `__tf32char_rd`, `__tf32char_rm`, `__tf32char_rn`, `__tf32bfloat16`, `__tf32bfloat16_tz`, `__tf32bfloat16_oz`, `__tf32bfloat16_up`, `__tf32bfloat16_dn`, `__tf32bfloat16_rd`, `__tf32bfloat16_rm`, `__tf32bfloat16_rn`, `__bfloat162float`, `__bfloat162float_tz`, `__bfloat162float_oz`, `__bfloat162float_up`, `__bfloat162float_dn`, `__bfloat162float_rd`, `__bfloat162float_rm`, `__bfloat162float_rn`, `__bfloat162half`, `__bfloat162half_tz`, `__bfloat162half_oz`, `__bfloat162half_up`, `__bfloat162half_dn`, `__bfloat162half_rd`, `__bfloat162half_rm`, `__bfloat162half_rn`, `__bfloat162short`, `__bfloat162short_tz`, `__bfloat162short_oz`, `__bfloat162short_up`, `__bfloat162short_dn`, `__bfloat162short_rd`, `__bfloat162short_rm`, `__bfloat162short_rn`, `__bfloat162int`, `__bfloat162int_tz`, `__bfloat162int_oz`, `__bfloat162int_up`, `__bfloat162int_dn`, `__bfloat162int_rd`, `__bfloat162int_rm`, `__bfloat162int_rn`, `__bfloat162char`, `__bfloat162char_tz`, `__bfloat162char_oz`, `__bfloat162char_up`, `__bfloat162char_dn`, `__bfloat162char_rd`, `__bfloat162char_rm`, `__bfloat162char_rn`, `__bfloat162tf32`, `__bfloat162tf32_tz`, `__bfloat162tf32_oz`, `__bfloat162tf32_up`, `__bfloat162tf32_dn`, `__bfloat162tf32_rd`, `__bfloat162tf32_rm`, `__bfloat162tf32_rn`, `__half2tf32`, `__half2tf32_tz`, `__half2tf32_oz`, `__half2tf32_up`, `__half2tf32_dn`, `__half2tf32_rd`, `__half2tf32_rm`, `__half2tf32_rn`, `__float2tf32`, `__float2tf32_tz`, `__float2tf32_oz`, `__float2tf32_up`, `__float2tf32_dn`, `__float2tf32_rd`, `__float2tf32_rm`, `__float2tf32_rn`, `__short2tf32`, `__short2tf32_tz`, `__short2tf32_oz`, `__short2tf32_up`, `__short2tf32_dn`, `__short2tf32_rd`, `__short2tf32_rm`, `__short2tf32_rn`, `__ushort2tf32`, `__ushort2tf32_tz`, `__ushort2tf32_oz`, `__ushort2tf32_up`, `__ushort2tf32_dn`, `__ushort2tf32_rd`, `__ushort2tf32_rm`, `__ushort2tf32_rn`, `__int2tf32`, `__int2tf32_tz`, `__int2tf32_oz`, `__int2tf32_up`, `__int2tf32_dn`, `__int2tf32_rd`, `__int2tf32_rm`, `__int2tf32_dn`, `__uint2tf32`, `__uint2tf32_tz`, `__uint2tf32_oz`, `__uint2tf32_up`, `__uint2tf32_dn`, `__uint2tf32_rd`, `__uint2tf32_rm`, `__uint2tf32_rn`, `__char2tf32`, `__uchar2tf32`, `__half2bfloat16`, `__half2bfloat16_tz`, `__half2bfloat16_oz`, `__half2bfloat16_up`, `__half2bfloat16_dn`, `__half2bfloat16_rd`, `__half2bfloat16_rm`, `__float2bfloat16`, `__float2bfloat16_tz`, `__float2bfloat16_oz`, `__float2bfloat16_up`, `__float2bfloat16_dn`, `__float2bfloat16_rd`, `__float2bfloat16_rm`, `__float2bfloat16_rn`, `__short2bfloat16`, `__short2bfloat16_tz`, `__short2bfloat16_oz`,

__short2bfloat16_up, __short2bfloat16_dn, __short2bfloat16_rd, __short2bfloat16_rm,
__short2bfloat16_rn, __ushort2bfloat16, __ushort2bfloat16_tz, __ushort2bfloat16_oz,
__ushort2bfloat16_up, __ushort2bfloat16_dn, __ushort2bfloat16_rd, __ushort2bfloat16_rm,
__ushort2bfloat16_rn, __int2bfloat16, __int2bfloat16_tz, __int2bfloat16_oz,
__int2bfloat16_up, __int2bfloat16_dn, __int2bfloat16_rd, __int2bfloat16_rm,
__int2bfloat16_rn, __uint2bfloat16, __uint2bfloat16_tz, __uint2bfloat16_oz,
__uint2bfloat16_up, __uint2bfloat16_dn, __uint2bfloat16_rd, __uint2bfloat16_rm,
__uint2bfloat16_rn, __char2bfloat16, __uchar2bfloat16, __bang_half2bfloat16,
__bang_half2bfloat16_tz, __bang_half2bfloat16_oz, __bang_half2bfloat16_up,
__bang_half2bfloat16_dn, __bang_half2bfloat16_rd, __bang_half2bfloat16_rm,
__bang_half2bfloat16_rn, __bang_bfloat162half, __bang_bfloat162half_tz,
__bang_bfloat162half_oz, __bang_bfloat162half_up, __bang_bfloat162half_dn,
__bang_bfloat162half_rd, __bang_int82bfloat16, __bang_int82bfloat16_tz,
__bang_int82bfloat16_up, __bang_int82bfloat16_dn, __bang_int82bfloat16_rn,
__bang_int82bfloat16_rm, __bang_bfloat162int8_tz, __bang_bfloat162int8_up,
__bang_bfloat162int8_dn, __bang_bfloat162int8_rn, __bang_bfloat162int8_oz,
__bang_bfloat162int8_rd, __bang_int162bfloat16, __bang_int162bfloat16_tz,
__bang_int162bfloat16_up, __bang_int162bfloat16_dn, __bang_int162bfloat16_rn,
__bang_int162bfloat16_rm, __bang_bfloat162int16_tz, __bang_bfloat162int16_oz,
__bang_bfloat162int16_up, __bang_bfloat162int16_dn, __bang_bfloat162int16_rn,
__bang_bfloat162int16_rm, __bang_int322bfloat16_tz, __bang_int322bfloat16_up,
__bang_int322bfloat16_dn, __bang_int322bfloat16_rn, __bang_int322bfloat16_oz,
__bang_bfloat162int32_tz, __bang_bfloat162int32_dn, __bang_bfloat162int32_rn,
__bang_bfloat162int32_oz, __bang_bfloat162int32_rd, __bang_bfloat162uchar,
__bang_bfloat162uchar_tz, __bang_bfloat162uchar_up, __bang_bfloat162uchar_dn,
__bang_bfloat162uchar_rn, __bang_bfloat162uchar_rm, __bang_float2uchar_tz,
__bang_float2uchar_up, __bang_float2uchar_dn, __bang_float2uchar_rn,
__bang_float2uchar_rm, __bang_uchar2int4_tz, __bang_uchar2int4_up,
__bang_uchar2int4_dn, __bang_uchar2int4_rd, __bang_uchar2int4_oz,
__bang_uchar2int4_rn, __bang_int42uchar, __bang_int42uchar_tz, __bang_int42uchar_oz,
__bang_int42uchar_up, __bang_int42uchar_dn, __bang_int42uchar_rn,
__bang_int42uchar_rm, __bang_uchar2int16_tz, __bang_uchar2int16_up,
__bang_uchar2int16_dn, __bang_uchar2int16_rn, __bang_uchar2int16_rd,
__bang_uchar2int16_oz, __bang_int162uchar, __bang_int162uchar_tz,
__bang_int162uchar_up, __bang_int162uchar_dn, __bang_int162uchar_rn,
__bang_uchar2int32, __bang_uchar2int32_tz, __bang_uchar2int32_up,
__bang_uchar2int32_dn, __bang_uchar2int32_rm, __bang_uchar2int32_oz,
__bang_uchar2int32_rd,

__bang_uchar2int32_rm, __bang_uchar2int32_rn, __bang_int322uchar,
 __bang_int322uchar_tz, __bang_int322uchar_oz, __bang_int322uchar_up,
 __bang_int322uchar_dn, __bang_int322uchar_rd, __bang_int322uchar_rm,
 __bang_int322uchar_rn, __bang_half2int32, __bang_half2int32_tz, __bang_half2int32_oz,
 __bang_half2int32_up, __bang_half2int32_dn, __bang_half2int32_rd,
 __bang_half2int32_rm, __bang_half2int32_rn, __bang_int322half, __bang_int322half_tz,
 __bang_int322half_oz, __bang_int322half_up, __bang_int322half_dh,
 __bang_int322half_rd, __bang_int322half_rm, __bang_int322half_rn, __bang_float2tf32,
 __bang_float2tf32_tz, __bang_float2tf32_oz, __bang_float2tf32_up, __bang_float2tf32_dn,
 __bang_float2tf32_rd, __bang_float2tf32_rm, __bang_float2tf32_rn, __bang_tf322float,
 __bang_tf322float_tz, __bang_tf322float_oz, __bang_tf322float_up, __bang_tf322float_dn,
 __bang_tf322float_rd, __bang_tf322float_rm, __bang_tf322float_rn, __bang_half2tf32,
 __bang_tf322half, __bang_tf322half_tz, __bang_tf322half_oz, __bang_tf322half_up,
 __bang_tf322half_dn, __bang_tf322half_rd, __bang_tf322half_rm, __bang_tf322half_rn,
 __bang_bfloat16tf32, __bang_tf322bfloat16, __bang_tf322bfloat16_tz,
 __bang_tf322bfloat16_oz, __bang_tf322bfloat16_up, __bang_tf322bfloat16_dn,
 __bang_tf322bfloat16_rd, __bang_tf322bfloat16_rm, __bang_tf322bfloat16_rn,
 __bang_int42tf32, __bang_int42tf32_tz, __bang_int42tf32_oz, __bang_int42tf32_up,
 __bang_int42tf32_dn, __bang_int42tf32_rd, __bang_int42tf32_rm, __bang_int42tf32_rn,
 __bang_tf322int4, __bang_tf322int4_tz, __bang_tf322int4_oz, __bang_tf322int4_up,
 __bang_tf322int4_dn, __bang_tf322int4_rd, __bang_tf322int4_rm, __bang_tf322int4_rn,
 __bang_int82tf32, __bang_int82tf32_tz, __bang_int82tf32_oz, __bang_int82tf32_up,
 __bang_int82tf32_dn, __bang_int82tf32_rd, __bang_int82tf32_rm, __bang_int82tf32_rn,
 __bang_tf322int8, __bang_tf322int8_tz, __bang_tf322int8_oz, __bang_tf322int8_up,
 __bang_tf322int8_dn, __bang_tf322int8_rd, __bang_tf322int8_rm, __bang_tf322int8_rn,
 __bang_int162tf32, __bang_int162tf32_tz, __bang_int162tf32_oz, __bang_int162tf32_up,
 __bang_int162tf32_dn, __bang_int162tf32_rd, __bang_int162tf32_rm, __bang_int162tf32_rn,
 __bang_int162tf32_rn, __bang_tf322int16, __bang_tf322int16_tz, __bang_tf322int16_oz,
 __bang_tf322int16_up, __bang_tf322int16_dn, __bang_tf322int16_rd,
 __bang_tf322int16_rm, __bang_tf322int16_rn, __bang_int322tf32, __bang_int322tf32_tz,
 __bang_int322tf32_oz, __bang_int322tf32_up, __bang_int322tf32_dn,
 __bang_int322tf32_rd, __bang_int322tf32_rm, __bang_int322tf32_rn, __bang_tf322int32,
 __bang_tf322int32_tz, __bang_tf322int32_oz, __bang_tf322int32_up,
 __bang_tf322int32_dn, __bang_tf322int32_rd, __bang_tf322int32_rm,
 __bang_tf322int32_rn, __bang_uchar2tf32, __bang_tf322uchar, __bang_tf322uchar_tz,
 __bang_tf322uchar_oz, __bang_tf322uchar_up, __bang_tf322uchar_dn,
 __bang_tf322uchar_rd, __bang_tf322uchar_rm, __bang_tf322uchar_rn, __ushort2half,
 __bang_sum, __bang_float2half_sr, __bang_float2tf32_sr, __bang_breduce,
 __bang_bexpand, __bang_discrete_atomic_add, __bang_discrete_atomic_and,
 __bang_discrete_atomic_cas, __bang_discrete_atomic_dec, __bang_discrete_atomic_exch,
 __bang_discrete_atomic_inc, __bang_discrete_atomic_max, __bang_discrete_atomic_min,
 __bang_discrete_atomic_or, __bang_discrete_atomic_xor, __bang_bexpand,
 __bang_breduce, __bang_conv_sparse, __bang_ssparse_filter_index,
 __bang_ssparse_filter_sparse_index, __bang_abs, __bang_conv_tf32,
 __bang_ssparse_filter_union_index, __bang_neu, __bang_geu,
 __bang_conv_partial_tf32, __bang_move, __bang_equ, __bang_neu, __bang_geu,
 __bang_gtu, __bang_leu, __bang_ltu, __bang_cycle_equ, __bang_cycle_neu,

`__bang_cycle_geu, __bang_cycle_gtu, __bang_cycle_leu, __bang_cycle_ltu,
__bang_equ_bitindex, __bang_neu_bitindex, __bang_geu_bitindex, __bang_gtu_bitindex,
__bang_leu_bitindex, __bang_ltu_bitindex, __bang_equ_scalar, __bang_neu_scalar,
__bang_geu_scalar, __bang_gtu_scalar, __bang_leu_scalar, __bang_ltu_scalar,
__bang_int322float, __bang_int322float_tz, __bang_int322float_oz, __bang_int322float_up,
__bang_int322float_dn, __bang_int322float_rd, __bang_int322float_rn,
__bang_int322float_rm, __bang_float2int32, __bang_float2int32_tz, __bang_float2int32_oz,
__bang_float2int32_up, __bang_float2int32_dn, __bang_float2int32_rd,
__bang_float2int32_rn, __bang_float2int32_rm, __bang_int82int32, __bang_int82int16,
__bang_int322int8, __bang_int322int16, __bang_int162int32, __bang_int162int8,
__bang_nearbyint.`

- Add `bfloat16_t` data type for `__bang_add`, `__bang_sub`, `__bang_mul`, `__bang_add_scalar`, `__bang_sub_scalar`, `__bang_cycle_add`, `__bang_cycle_sub`, `__bang_cycle_mul`, `__bang_count`, `__bang_count_bitindex`, `__bang_square`, `__bang_findfirst1`, `__bang_findlast1`, `__bang_relu`, `__bang_relun`, `__bang_eq_scalar`, `__bang_ne_scalar`, `__bang_le_scalar`, `__bang_lt_scalar`, `__bang_ge_scalar`, `__bang_gt_scalar`, `__bang_and_scalar`, `__bang_or_scalar`, `__bang_xor_scalar`, `__bang_and`, `__bang_or`, `__bang_not`, `__bang_xor`, `__bang_cycle_and`, `__bang_cycle_or`, `__bang_cycle_xor`, `__bang_band_scalar`, `__bang_bor_scalar`, `__bang_bxor_scalar`, `__bang_fusion`, `__bang fabsmax`, `__bang fabsmin`, `__bang fcmpfilter`, `__bang_eq`, `__bang_ne`, `__bang_gt`, `__bang_ge`, `__bang_le`, `__bang_lt`, `__bang_eq_bitindex`, `__bang_ne_bitindex`, `__bang_lt_bitindex`, `__bang_le_bitindex`, `__bang_gt_bitindex`, `__bang_ge_bitindex`, `__bang_max`, `__bang_min`, `__bang_argmax`, `__bang_argmin`, `__bang_maxequal`, `__bang_minequal`, `__bang_cycle_maxequal`, `__bang_cycle_minequal`, `__bang_cycle_eq`, `__bang_cycle_ne`, `__bang_cycle_lt`, `__bang_cycle_le`, `__bang_cycle_gt`, `__bang_cycle_ge`, `__bang_sumpool`, `__bang_unpool`, `__bang_maxpool`, `__bang_maxpool_index`, `__bang_maxpool_value_index`, `__bang_minpool`, `__bang_minpool_index`, `__bang_minpool_value_index`, `__bang_avgpool`, `__bang_avgpool_bp`, `__bang_maxpool_bp`, `__bang_atomic_add`, `__bang_atomic_max`, `__bang_atomic_min`, `__bang_atomic_exch`, `__bang_atomic_reduce_max`, `__bang_atomic_reduce_min`, `__bang_atomic_reduce_cas`, `__bang_atomic_reduce_exch`, `__bang_atomic_add`, `__bang_atomic_max`, `__bang_atomic_min`, `__bang_atomic_exch`, `__bang_atomic_reduce_add`, `__bang_atomic_reduce_max`, `__bang_atomic_reduce_min`, `__bang_atomic_reduce_exch`, `__bang_write_value`, `__memset_nram`, `__bang_write_value`, `__nramset`, `__bang_select`, `__bang_select_bitindex`, `__bang_collect`, `__bang_collect_bitindex`.
- Add return value for `__bang_count`, `__bang_count_bitindex`, `__bang_findfirst1`, `__bang_findlast1`.
- Add void for `__bang_maskmove`.
- Deprecate built-in functions including `__bang_fix82half`, `__bang_add_const`, `__bang_sub_const`, `__bang_mul_const`, `__bang_ge_const`, `__bang_bor`, `__bang_band`, `__bang_bxor`, `__bang_bnot`, `__bang_cycle_bxor`, `__bang_cycle_bor`, `__bang_cycle_band`, `__bang_maskmove_bitindex`, `__bang_maskmove`, `__bang_count`, `__bang_count_bitindex`, `__bang_findfirst1`, `__bang_findlast1`, `__bang_maximum`, `__bang_lock`, `__bang_unlock`, `__nramset_float`, `__nramset_half`, `__nramset_int`, `__nramset_unsigned_int`, `__nramset_short`, `__nramset_unsigned_short`, `__nramset`, `__bang_div`, `__bang_half2uchar_dn`, `__bang_pad`, `__bang_int2float`, `__bang_float2int_dn`, `__bang_float2int_tz`, `__bang_float2int_oz`, `__bang_float2int_up`, `__bang_float2int_rd`,

`__bang_float2int_rn`, `__bang_float2int_rm`, `__bang_char2int`, `__bang_char2short`,
`__bang_int2char`, `__bang_int2short`, `__bang_short2char`, `__bang_short2int`,
`__bang_half2char_dn`.

- **V3.8.0**

Date : December 20, 2021

Changes :

- Add built-in functions including `__bang_abs`, `__bang_square`, `__bang_relu`, `__bang_sra`, `__bang_sll`, `__bang_srl`, `__bang_neg`, `__bang_nsa`, `__bang_ror`, `__bang_rol`, `__nramset`, `__sramset`, `__ldramset`, `__gdramset`, `__bang_adds`, `__bang_subs`, `__bang_muls`, `__bang_mulh`, `__bang_set0in32`, `__bang_set1in32`, `__bang_adds_scalar`, `__bang_subs_scalar`, `__bang_muls_scalar`, `__bang_mulh_scalar`, `__bang_cycle_adds`, `__bang_cycle_subs`, `__bang_cycle_muls`, `__bang_cycle_mulh`.
- Add `int8`, `int16` and `int32` data types for `__bang_or`, `__bang_and`, `__bang_xor`, `__bang_not`, `__bang_band`, `__bang_bor`, `__bang_bxor`, `__bang_bnot`.

- **V3.7.0**

Date : December 15, 2021

Changes :

- Add built-in functions: [Atomic Reduce Functions](#).
- Add `int4` and `fix4` date types for `__bang_conv`, `__bang_conv_partial`.
- Add `int4` date type for [Vector Type Conversion Functions](#).
- Support GDRAM2GDRAM and GDRAM2LDRAM direction in `__memcpy_async` interface.
- Support negative `src_stride` for 2D `memcpy`, and support negative `dim0_stride` for 3D `memcpy`.

- **V3.6.0**

Cambricon@155chb

Update Data : November 18, 2021

Changes :

- Internal update.

- **V3.5.0**

Update Data : August 30, 2021

Changes :

- Internal update.

- **V3.4.0**

Date : July 20, 2021

Changes :

- Support GDRAM2WRAM direction in `__memcpy_async` interface.
- Add built-in functions: `__bang_rem`, `__is_nram`, `__is_wram`, `__is_sram`, `__is_ipu`, `__is_mpu`, `__popcnt`, `__bang_popcnt`.

- **V3.3.0**

Date : June 26, 2021

Changes :

- Support `tp_322` architecture.
- Add built-in functions on all targets: `__sync_all_ipu`, `__sync_all_mpu`, `__sync_cluster`, `__sync_all`.
- Add `int16`, `half` and `float` data types for built-in function: `__bang_histogram`.

- **V3.2.0**

Date : April 26, 2021

Changes :

- Add built-in functions: `__memcpy_async`, `__bang_write_value`, `__bang_integral`, `__bang_lut_s16`, `__bang_lut_s32`.
- Add `<indilation_x>`, `<indilation_y>`, `<outdilation_x>` and `<outdilation_y>` parameters for built-in functions `__bang_conv` or `__bang_conv_partial`.
- Add built-in functions on `(m)tp_3xx` : `__bang_taylor3_sin`, `__bang_taylor4_sin`, `__bang_taylor3_cos`, `__bang_taylor4_cos`, `__bang_taylor3_tanh`, `__bang_taylor4_tanh`, `__bang_taylor3_sigmoid`, `__bang_taylor4_sigmoid`, `__bang_taylor3_softplus`, `__bang_taylor4_softplus`.

• V3.1.0

Date : March 24, 2021

Changes :

- Add built-in functions: `__bang_argmax`, `__bang_argmin`.
- Add built-in functions on `(m)tp_3xx` : `__bang_active_abs`, `__bang_active_cos`, `__bang_active_exp`, `__bang_active_exphp`, `__bang_active_exp_less_0`, `__bang_active_gelu`, `__bang_active_gelup`, `__bang_active_log`, `__bang_active_loghp`, `__bang_active_pow2`, `__bang_active_recip`, `__bang_active_recip_greater_1`, `__bang_active_reciphp`, `__bang_active_relu`, `__bang_active_rsqrt`, `__bang_active_rsrthp`, `__bang_active_sigmoid`, `__bang_active_sign`, `__bang_active_sin`, `__bang_active_sqrt`, `__bang_active_sqrthp`, `__bang_active_tanh`, `__bang_rand`.
- Add `bfloat16_t` data type for built-in functions: `__bang_conv`, `__bang_conv_partial`, `__bang_float2bfloat16_rm`, `__bang_float2bfloat16_rn`, `__bang_float2bfloat16_rd`, `__bang_float2bfloat16_up`, `__bang_float2bfloat16_oz`, `__bang_float2bfloat16_dn`, `__bang_float2bfloat16_tz`, `__bang_bfloat162float`.

• V3.0.0

Cambricon@155chb

Date : March 8, 2021

Changes :

- Add built-in functions supported on `mtp_372` : `__bang_sin`, `__bang_cos`, `__bang_log`, `__bang_pow2`, `__bang_sqrt`, `__bang_recip`, `__bang_rsqrt`, `__bang_fusion`, `__bang_fcmpfilter`, `__bang fabsmax`, `__bang fabsmin`, `__bang_relu`, `__bang_relun`, `__bang_int2float`, `__memcpy`, `__bang_float2half_rm`, `__bang_float2half_rn`, `__bang_float2int16_rm`, `__bang_float2int16_rn`, `__bang_float2int8_rm`, `__bang_float2int8_rn`, `__bang_half2int16_rm`, `__bang_half2int16_rn`, `__bang_half2int8_rm`, `__bang_half2int8_rn`, `__bang_half2short_rm`, `__bang_half2short_rn`, `__bang_float2int_dn`, `__bang_float2int_tz`, `__bang_float2int_oz`, `__bang_float2int_up`, `__bang_float2int_rd`, `__bang_float2int_rn`, `__bang_float2int_rm`, `__bang_maxpool_value_index`, `__bang_minpool_value_index`.

• V2.16.0

Date : March 2, 2021

Changes :

- Support `__memcpy_async` with stride.

• V2.15.0

Date : February 3, 2021

Changes :

- Add new direction NRAM2WRAM for `__memcpy` with stride.
- Update input dilation parameter for `__bang_conv_partial` on `mtp_270` and `mtp_290`.

Date : January 21, 2021

Changes :

- Add build-in function: `__memset_nram`.

- **V2.8.0**

Date : August 11, 2020

Changes :

- Add build-in function: `__bang_ge_const`, `__bang_taylor3_sin`, `__bang_taylor4_sin`, `__bang_taylor3_cos`, `__bang_taylor4_cos`, `__bang_taylor3_tanh`, `__bang_taylor4_tanh`, `__bang_taylor3_sigmoid`, `__bang_taylor4_sigmoid`, `__bang_taylor3_softplus`, `__bang_taylor4_softplus`.
- Add example for `__memcpy`.
- Delete <fix_position> of `__bang_uchar2half`.
- Update example of `__bang_maxpool_bp`.
- Update align of `__bang_mul`, `__bang_sub`, `__bang_add`, `__bang_mul_const`, `__bang_rotate90`, `__bang_rotate180`, `__bang_rotate270`.
- Update parameters of `__bang_avgpool_bp`, `__bang_maxpool_bp`, `__bang_histogram`, `__bang_mirror`.
- Update homologous operand of `__bang_mul`, `__bang_add_const`, `__bang_avgpool`, `__bang_collect`, `__bang_collect_bitindex`, `__bang_conv`, `__bang_conv_partial`, `__bang_count`, `__bang_count_bitindex`, `__bang_cycle_add`, `__bang_cycle_ne`, `__bang_cycle_le`, `__bang_cycle_gt`, `__bang_cycle_mul`, `__bang_cycle_sub`, `__bang_div`, `__bang_findfirst1`, `__bang_findlast1`, `__bang_count`, `__bang_maskmove_bitindex`, `__bang_maximum`, `__bang_maxpool`, `__bang_maxpool_index`, `__bang_minpool_index`, `__bang_mlp`, `__bang_mul_const`, `__bang_pad`, `__bang_reduce_sum`, `__bang_select`, `__bang_select_bitindex`, `__bang_square`, `__bang_sub_const`, `__bang_smpool`, `__bang_tiling_2d_b128`, `__bang_tiling_2d_b16`, `__bang_tiling_2d_b256`, `__bang_tiling_2d_b32`, `__bang_tiling_2d_b64`, `__bang_tiling_2d_b8`, `__bang_tiling_3d_b1024`, `__bang_transpose`, `__bang_unpool`, `__bang_reshape_filter`, `__bang_reshape_nhwc2nchw`, `__bang_reshape_nchw2nhwc`, `__bang_cycle_eq`, `__bang_eq_bitindex`, `__bang_ge`, `__bang_ge_bitindex`, `__bang_gt`, `__bang_gt_bitindex`, `__bang_le`, `__bang_le_bitindex`, `__bang_lt`, `__bang_lt_bitindex`, `__bang_ne`, `__bang_ne_bitindex`, `__bang_and`, `__bang_band`, `__bang_bnot`, `__bang_bor`, `__bang_bxor`, `__bang_cycle_and`, `__bang_cycle_band`, `__bang_cycle_bor`, `__bang_cycle_bxor`, `__bang_cycle_maxequal`, `__bang_cycle_minequal`, `__bang_cycle_or`, `__bang_cycle_xor`, `__bang_max`, `__bang_maxequal`, `__bang_min`, `__bang_minequal`, `__bang_or`, `__bang_xor`.

- **V2.6.0**

Date : May 27, 2020

Changes :

- Update high precision active interface.



3 Built-in Functions

3.1 1D Memcpy Functions

3.1.1 __bang_move

```
void __bang_move(void *dst,
                 const void *src,
                 int size)
```

Copies `<size>` bytes data from source address `<src>` to destination address `<dst>`.

Parameters

- [out] `dst`: The address of destination area.
- [in] `src`: The address of source area.
- [in] `size`: The number of bytes to be copied.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero;
- `<size>` must be divisible by 128 on `(m)tp_2xx` ;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx` .
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(int8_t *dst, int8_t *src) {
    __nram__ int8_t src1_nram[512];
    __nram__ int8_t src2_nram[512];
    __memcpy(src1_nram, src, 512, GDRAM2NRAM);
    __bang_move(src2_nram, src1_nram, 512);
```

```

    __memcpy(dst, src2_nram, 512, NRAM2GDRAM);
}

```

3.1.2 `__memcpy`

```

void __memcpy(void *dst,
              const void *src,
              int size,
              mluMemcpyDirection_t dir)

```

```

void __memcpy(void *dst,
              const void *src,
              int size,
              mluMemcpyDirection_t dir,
              int id_dst_cluster)

```

Copies `<size>` bytes data from source address `<src>` to destination address `<dst>`. The copy direction is specified by `<dir>`.

Parameters

- [out] `dst`: The address of destination area.
- [in] `src`: The address of source area.
- [in] `size`: The number of bytes to be copied.
- [in] `dir`: Copy direction.
- [in] `id_dst_cluster`: Destination cluster ID.

Return

- `void`.

Remark

- `<size>` must be greater than zero;
- `__memcpy` with `<id_dst_cluster>` is not supported on `(m)tp_5xx` or higher;
- The address of `__nram__` address space, to which `<src>` or `<dst>` points, must be 64-byte aligned, when `<dir>` is NRAM2NRAM on `(m)tp_2xx` or higher;
- `<id_dst_cluster>` is necessarily used when `<dir>` is SRAM2SRAM and data is copied across different clusters. When there is no `<id_dst_cluster>`, it means copy within cluster;
- If the size of the vector with the address space of WRAM is not 8-byte aligned on each LT, the value of the unaligned part is uncertain on `tp_322` and `(m)tp_372`, and is 0 on other targets;
- The alignment constraints of address of `<dst>` or `<src>` in `__memcpy` are shown in the table [Alignment Constraints of Address Space in `__memcpy`](#);

Table 3.1: Alignment Constraints of Address Space in `__memcpy`

Address Space	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM	1B	1B	1B	1B	1B
LDRAM	1B	1B	1B	1B	1B
SRAM	NA	1B	NA	1B	1B
NRAM	1B	1B	1B	1B	1B
WRAM	8B	8B	8B	8B	8B

- The supported copy directions of `__memcpy` on different targets are shown in the table [Direction Constraints of `__memcpy`](#);

Table 3.2: Direction Constraints of `__memcpy`

Direction	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM2GDRAM	Y	Y	Y	Y	Y
GDRAM2LDRAM	Y	Y	Y	Y	Y
GDRAM2SRAM	N	Y	N	Y	Y
GDRAM2NRAM	Y	Y	Y	Y	Y
GDRAM2WRAM	Y	Y	Y	Y	Y
LDRAM2GDRAM	Y	Y	Y	Y	Y
LDRAM2LDRAM	N	N	N	N	N
LDRAM2SRAM	N	Y	N	Y	Y
LDRAM2NRAM	Y	Y	Y	Y	Y
LDRAM2WRAM	Y	Y	Y	Y	Y
SRAM2GDRAM	N	Y	N	Y	Y
SRAM2LDRAM	N	Y	N	Y	Y
SRAM2SRAM	N	Y	N	Y	Y
SRAM2NRAM	N	Y	N	Y	Y
SRAM2WRAM	N	Y	N	Y	Y
NRAM2GDRAM	Y	Y	Y	Y	Y

continues on next page

Table 3.2 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
NRAM2LDRAM	Y	Y	Y	Y	Y
NRAM2SRAM	N	Y	N	Y	Y
NRAM2NRAM	Y	Y	Y	Y	Y
NRAM2WRAM	Y	Y	Y	Y	Y
WRAM2GDRAM	Y	Y	Y	Y	Y
WRAM2LDRAM	Y	Y	Y	Y	Y
WRAM2SRAM	N	Y	N	Y	Y
WRAM2NRAM	Y	Y	Y	Y	Y
WRAM2WRAM	N	N	N	N	N

- The alignment constraints of <size> in __memcpy are shown in the table [Alignment Constraints of size in __memcpy](#);

Table 3.3: Alignment Constraints of size in __memcpy

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2GDRAM	1B	1B	1B	1B	1B
GDRAM2LDRAM	1B	1B	1B	1B	1B
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	64B	64B	16B	16B	16B
LDRAM2GDRAM	1B	1B	1B	1B	1B
LDRAM2LDRAM	NA	NA	NA	NA	NA
LDRAM2SRAM	NA	1B	NA	1B	1B
LDRAM2NRAM	1B	1B	1B	1B	1B
LDRAM2WRAM	64B	64B	16B	16B	16B
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	1B	NA	1B	1B

continues on next page

Table 3.3 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
SRAM2SRAM	NA	1B	NA	1B	1B
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	64B	NA	16B	16B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	1B	1B	1B	1B	1B
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	128B	128B	1B	1B	1B
NRAM2WRAM	64B	64B	16B	16B	16B
WRAM2GDRAM	64B	64B	16B	16B	16B
WRAM2LDRAM	64B	64B	16B	16B	16B
WRAM2SRAM	NA	64B	NA	16B	16B
WRAM2NRAM	64B	64B	16B	16B	16B
WRAM2WRAM	NA	NA	NA	NA	NA

Instruction Pipeline

- Execute in IO instruction pipeline, if either <src> or <dst> is off-chip address;
- Execute in Compute instruction pipeline in NRAM2NRAM direction, if both <src> and <dst> are in-chip address;
- Execute in Move instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```

if (clusterId == 0) {
    // Step1. load data from GDRAM to cluster0's SRAM
    // auto running on MPU, no need to specify coreId == 0x80
    __memcpy(S, src, size * coreDim, GDRAM2SRAM);
    __sync_cluster();

    // Step2. test inside cluster0
    __memcpy(A, S + offset, size, SRAM2NRAM);
    __memcpy(C, A, size, NRAM2NRAM);
    __memcpy(S + offset, C, size, NRAM2SRAM);
}

```

```

__sync_cluster();

// Step3. push S from Cluster0 to Cluster1
__memcpy(S, S, size * coreDim, SRAM2SRAM, clusterId + 1);
}

```

3.1.3 __memcpy_async

```

void __memcpy_async(void *dst,
                   const void *src,
                   int size,
                   mluMemcpyDirection_t dir)

```

Copies `<size>` bytes data from source address `<src>` to destination address `<dst>` asynchronously. The copy direction is specified by `<dir>`.

Parameters

- [out] `dst`: The address of destination area.
- [in] `src`: The address of source area.
- [in] `size`: The number of bytes to be copied.
- [in] `dir`: Copy direction.

Return

- `void`.

Remark

Cambricon@155chb

- `<size>` must be a greater than zero;
- `__memcpy_async` with `<id_dst_cluster>` is not supported on `(m)tp_5xx` or higher;
- If the size of the vector with the address space of WRAM is not 8-byte aligned on each LT, the value of the unaligned part is uncertain on `tp_322` and `(m)tp_372`, and is 0 on other targets;
- The alignment constraints of address of `<dst>` or `<src>` in `__memcpy_async` are shown in the table [Alignment Constraints of Address Space in __memcpy](#);
- The supported copy directions of `__memcpy_async` on different targets are shown in the table [Direction Constraints of __memcpy_async](#);

Table 3.4: Direction Constraints of `__memcpy_async`

Direction	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM2GDRAM	Y	Y	Y	Y	Y
GDRAM2LDRAM	Y	Y	Y	Y	Y
GDRAM2SRAM	N	Y	N	Y	Y
GDRAM2NRAM	Y	Y	Y	Y	Y

continues on next page

Table 3.4 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2WRAM	Y	Y	Y	Y	Y
LDRAM2GDRAM	N	N	N	N	N
LDRAM2LDRAM	N	N	N	N	N
LDRAM2SRAM	N	N	N	N	N
LDRAM2NRAM	N	N	N	N	N
LDRAM2WRAM	N	N	N	N	N
SRAM2GDRAM	N	Y	N	Y	Y
SRAM2LDRAM	N	N	N	N	N
SRAM2SRAM	N	Y	N	Y	Y
SRAM2NRAM	N	Y	N	Y	Y
SRAM2WRAM	N	Y	N	Y	Y
NRAM2GDRAM	Y	Y	Y	Y	Y
NRAM2LDRAM	N	N	N	N	N
NRAM2SRAM	N	Y	N	Y	Y
NRAM2NRAM	Y	Y	Y	Y	Y
NRAM2WRAM	Y	Y	Y	Y	Y
WRAM2GDRAM	N	N	N	N	N
WRAM2LDRAM	N	N	N	N	N
WRAM2SRAM	N	Y	N	Y	Y
WRAM2NRAM	Y	Y	Y	Y	Y
WRAM2WRAM	N	N	N	N	N

- The alignment constraints of <size> in __memcpy_async are shown in the table [Alignment Constraints of size in __memcpy_async](#);

Table 3.5: Alignment Constraints of size in __memcpy_async

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2GDRAM	NA	NA	NA	NA	NA
GDRAM2LDRAM	NA	NA	NA	NA	NA
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	64B	64B	16B	16B	16B
LDRAM2GDRAM	NA	NA	NA	NA	NA
LDRAM2LDRAM	NA	NA	NA	NA	NA
LDRAM2SRAM	NA	NA	NA	NA	NA
LDRAM2NRAM	NA	NA	NA	NA	NA
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	NA	NA	NA	NA
SRAM2SRAM	NA	NA	NA	NA	NA
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	64B	NA	16B	16B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	NA	NA	NA	NA	NA
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	128B	128B	1B	1B	1B
NRAM2WRAM	64B	64B	16B	16B	16B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	64B	NA	16B	16B
WRAM2NRAM	64B	64B	16B	16B	16B
WRAM2WRAM	NA	NA	NA	NA	NA

Instruction Pipeline

- Execute in IO instruction pipeline, if either <src> or <dst> is off-chip address;
- Execute in Compute instruction pipeline in NRAM2NRAM direction, if both <src> and

- <dst> are in-chip address;
- Execute in Move instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.2 1D Memset Functions

3.2.1 __bang_write_value

```
void __bang_write_value(void *dst,
                      int elem_count,
                      char value)

void __bang_write_value(void *dst,
                      int elem_count,
                      int value)

void __bang_write_value(void *dst,
                      int elem_count,
                      short value)

void __bang_write_value(void *dst,
                      int elem_count,
                      bfloat16_t value)

void __bang_write_value(void *dst,
                      int elem_count,
                      float value)

void __bang_write_value(void *dst,
                      int elem_count,
                      half value)
```

Sets a vector in the __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: The value to be set to <dst>.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> must point to __nram__ address space;
- The address of <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(<value>) must be divisible by 64 (m)tp_2xx;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200 ;
- CNCC Version: cncc --version >= 3.2.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx .

Example

- None.

3.2.2 __bang_write_zero

```
void __bang_write_zero(signed char *dst,
                      int elem_count)
```

```
void __bang_write_zero(unsigned char *dst,
                      int elem_count)
```

```
void __bang_write_zero(short *dst,
                      int elem_count)
```

```
void __bang_write_zero(unsigned short *dst,
                      int elem_count)
```

```
void __bang_write_zero(int *dst,
                      int elem_count)
```

```
void __bang_write_zero(unsigned int *dst,
                      int elem_count)
```

```
void __bang_write_zero(half *dst,
                      int elem_count)
```

```
void __bang_write_zero(float *dst,
                      int elem_count)
```

Sets a vector in __nram__ address space pointed by <dst> to zero.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements in destination vector.

Return

- void.

Remark

- <dst> must point to __nram__ address space;
- <elem_count> must be greater than zero;
- The address of <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.2.3 __gdramset

```
void __gdramset(void *dst,
                 int elem_count,
                 char value)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 short value)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 bfloat16_t value)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 half value)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 float value)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 int value)
```

Sets a vector in __mlu_device__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.

- [in] `value`: Value to be set.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` must point to `__mlu_device__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.2.4 `__lDRAMset`

```
void __lDRAMset(void *dst,
                 int elem_count,
                 char value)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 short value)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 bfloat16_t value)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 half value)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 float value)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 int value)
```

Sets a vector in `__lDRAM__` address space pointed by `<dst>` to the specified `<value>`.

Parameters

- [out] `dst`: The address of destination vector.

- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> must point to __lram__ address space;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.2.5 __memset_nram

```
void __memset_nram(void *dst,
                   int elem_count,
                   char value)
```

```
void __memset_nram(void *dst,
                   int elem_count,
                   short value)
```

```
void __memset_nram(void *dst,
                   int elem_count,
                   half value)
```

```
void __memset_nram(void *dst,
                   int elem_count,
                   bfloat16_t value)
```

```
void __memset_nram(void *dst,
                   int elem_count,
                   float value)
```

```
void __memset_nram(void *dst,
                   int elem_count,
                   int value)
```

Sets a vector in __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `elem_count`: The number of elements to be set.
- [in] `value`: Value to be set.

Return

- `void`.

Remark

- `<dst>` must point to `__nram__` address space;
- `<elem_count>` must be greater than zero;
- `bfloat16_t` is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Move.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.16.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.2.6 `__sramset`

```
void __sramset(void *dst,
               int elem_count,
               char value)

void __sramset(void *dst,
               int elem_count,
               half value)

void __sramset(void *dst,
               int elem_count,
               bfloat16_t value)

void __sramset(void *dst,
               int elem_count,
               short value)

void __sramset(void *dst,
               int elem_count,
               float value)

void __sramset(void *dst,
               int elem_count,
               int value)
```

Sets a vector in `__mlu_shared__` address space pointed by `<dst>` to the specified `<value>`.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> must point to __mlu_shared__ address space;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Move.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.3 2D Memcpy Functions

3.3.1 __bang_move

```
void __bang_move(void *dst,
                 const void *src,
                 int size,
                 int dst_stride,
                 int src_stride,
                 int segnum)
```

Copies <size> bytes data from <src> source address to <dst> destination address.

As shown in Figure [Memory Copy Function with Stride](#), the blue background indicates the data to be copied.

Parameters

- [out] dst: The address of destination area.
- [in] src: The address of source area.
- [in] size: The number of bytes of one data segment.
- [in] dst_stride: The destination stride in bytes.
- [in] src_stride: The source stride in bytes.
- [in] segnum: The number of data segment minus one.

Return

- void.

Remark

- When <segnum> is 0, it means that it is copied once. The real segment number is the

- given value plus one;
- <src> and <dst> must point to __nram__ address space;
 - <size> must be greater than zero;
 - <size> must be divisible by 128 on (m)tp_2xx and mtp_372;
 - <dst_stride> must be greater than or equal to <size>;
 - <dst_stride> must be divisible by 64 on (m)tp_2xx;
 - <src_stride> must be greater than or equal to zero;
 - <src_stride> must be divisible by 64 on (m)tp_2xx;
 - <segnr> must be equal to or greater than 0;
 - <dst> cannot be overlapped with <src> on (m)tp_2xx and mtp_370.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Instruction Pipeline

- Compute.

Example

```
#include <bang.h>

#define LEN 64
#define DST_STRIDE 512
#define SRC_STRIDE 128
#define SEGNUM 7

__mlu_entry__ void kernel2D(int8_t *dst, int8_t *src) {
    __nram__ int8_t src1_nram[512];
    __nram__ int8_t src2_nram[512];
    __memcpy(src1_nram, src, 512, GDRAM2NRAM);
    __bang_move(src2_nram, src1_nram, LEN, DST_STRIDE, SRC_STRIDE, SEGNUM);
    __memcpy(dst, src2_nram, 512, NRAM2GDRAM);
}
```

3.3.2 __memcpy

```
void __memcpy(void *dst,
             const void *src,
             int size,
             mluMemcpyDirection_t dir,
             int dst_stride,
             int src_stride,
             int segnum)
```

```
void __memcpy(void *dst,
              const void *src,
              int size,
              mluMemcpyDirection_t dir,
              int dst_stride,
              int src_stride,
              int segnum,
              int id_dst_cluster)
```

Copies `<size>` bytes data from source address `<src>` to destination address `<dst>`. The copy direction is specified by `<dir>`.

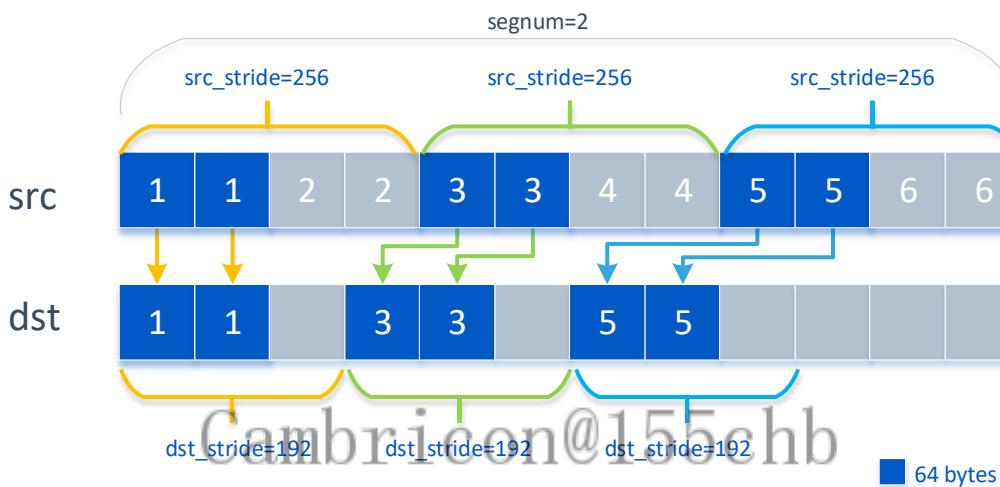


Fig. 3.1: Memory Copy Function with Stride

As shown in Figure [Memory Copy Function with Stride](#), each cell represents 64 bytes and the cells with blue background indicates the data to be copied which is 128 bytes. There are 3 segments in source area and in each segment only the first 128 bytes of every 256 bytes are copied to destination area. Please note that `<segnum>` is the real segment number minus one, that is, `<segnum>` = 2 in this case.

Parameters

- [out] `dst`: The address of destination area.
- [in] `src`: The address of source area.
- [in] `size`: The number of bytes of one segment.
- [in] `dir`: Copy direction.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.
- [in] `id_dst_cluster`: Destination cluster ID.

Return

- void.

Remark

- <size> must be greater than zero;
- The range of <segnum> is [0, 65535];
- When the segment number is 0, it means that it is copied once. The real segment number value is the given value plus one;
- <dst_stride> must be greater than or equal to <size>;
- __memcpy with <id_dst_cluster> is not supported on (m)tp_5xx or higher;
- <id_dst_cluster> is necessarily used when <dir> is SRAM2SRAM and data is copied across different clusters. When there is no <id_dst_cluster>, it means copy within cluster;
- If the size of the vector with the address space of WRAM is not 8-byte aligned on each LT, the value of the unaligned part is uncertain on tp_322 and (m)tp_372, and is 0 on other targets;
- The alignment constraints of address of <dst> or <src> in __memcpy are shown in the table [Alignment Constraints of Address Space in __memcpy](#);

Table 3.6: Alignment Constraints of Address Space in __memcpy

Address Space	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM	1B	1B	1B	1B	1B
LDRAM	1B	1B	1B	1B	1B
SRAM	NA	1B	NA	1B	1B
NRAM	1B	1B	1B	1B	1B
WRAM	8B	8B	8B	8B	8B

- The supported copy directions of __memcpy on different targets are shown in the table [Direction Constraints of __memcpy](#);

Table 3.7: Direction Constraints of __memcpy

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2GDRAM	Y	Y	Y	Y	Y
GDRAM2LDRAM	Y	Y	Y	Y	Y
GDRAM2SRAM	N	Y	N	Y	Y
GDRAM2NRAM	Y	Y	Y	Y	Y
GDRAM2WRAM	Y	Y	Y	Y	Y

continues on next page

Table 3.7 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
LDRAM2GDRAM	Y	Y	Y	Y	Y
LDRAM2LDRAM	Y	Y	Y	Y	Y
LDRAM2SRAM	N	Y	N	Y	Y
LDRAM2NRAM	Y	Y	Y	Y	Y
LDRAM2WRAM	N	N	N	N	N
SRAM2GDRAM	N	Y	N	Y	Y
SRAM2LDRAM	N	Y	N	Y	Y
SRAM2SRAM	N	Y	N	Y	Y
SRAM2NRAM	N	Y	N	Y	Y
SRAM2WRAM	N	Y	N	Y	Y
NRAM2GDRAM	Y	Y	Y	Y	Y
NRAM2LDRAM	Y	Y	Y	Y	Y
NRAM2SRAM	N	Y	N	Y	Y
NRAM2NRAM	Y	Y	Y	Y	Y
NRAM2WRAM	Y	Y	Y	Y	Y
WRAM2GDRAM	N	N	N	N	N
WRAM2LDRAM	N	N	N	N	N
WRAM2SRAM	N	N	N	N	N
WRAM2NRAM	N	N	N	N	N
WRAM2WRAM	N	N	N	N	N

- The alignment constraints of <size> in __memcpy are shown in the table [Alignment Constraints of size in __memcpy](#).

Table 3.8: Alignment Constraints of `size` in `__memcpy`

Direction	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM2GDRAM	1B	1B	1B	1B	1B
GDRAM2LDRAM	1B	1B	1B	1B	1B
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	1B	1B	1B	1B	1B
LDRAM2GDRAM	1B	1B	1B	1B	1B
LDRAM2LDRAM	1B	1B	1B	1B	1B
LDRAM2SRAM	NA	1B	NA	1B	1B
LDRAM2NRAM	1B	1B	1B	1B	1B
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	1B	NA	1B	1B
SRAM2SRAM	NA	1B	NA	1B	1B
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	1B	NA	1B	1B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	1B	1B	1B	1B	1B
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	1B	1B	1B	1B	1B
NRAM2WRAM	1B	1B	1B	1B	1B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	NA	NA	NA	NA
WRAM2NRAM	NA	NA	NA	NA	NA
WRAM2WRAM	NA	NA	NA	NA	NA

- The alignment constraints of `<dst_stride>` in `__memcpy` are shown in the table [Alignment Constraints of `dst_stride` in `__memcpy`](#);

Table 3.9: Alignment Constraints of `dst_stride` in `__memcpy`

Direction	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM2GDRAM	1B	1B	1B	1B	1B
GDRAM2LDRAM	1B	1B	1B	1B	1B
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	8B	8B	8B	8B	8B
LDRAM2GDRAM	1B	1B	1B	1B	1B
LDRAM2LDRAM	1B	1B	1B	1B	1B
LDRAM2SRAM	NA	1B	NA	1B	1B
LDRAM2NRAM	1B	1B	1B	1B	1B
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	1B	NA	1B	1B
SRAM2SRAM	NA	1B	NA	1B	1B
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	8B	NA	8B	8B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	1B	1B	1B	1B	1B
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	1B	1B	1B	1B	1B
NRAM2WRAM	8B	8B	8B	8B	8B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	NA	NA	NA	NA
WRAM2NRAM	NA	NA	NA	NA	NA
WRAM2WRAM	NA	NA	NA	NA	NA

- The alignment constraints of `<src_stride>` in `__memcpy` are shown in the table [Alignment Constraints of `src_stride` in `__memcpy`](#).

Table 3.10: Alignment Constraints of `src_stride` in `__memcpy`

Direction	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM2GDRAM	1B	1B	1B	1B	1B
GDRAM2LDRAM	1B	1B	1B	1B	1B
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	1B	1B	1B	1B	1B
LDRAM2GDRAM	1B	1B	1B	1B	1B
LDRAM2LDRAM	1B	1B	1B	1B	1B
LDRAM2SRAM	NA	1B	NA	1B	1B
LDRAM2NRAM	1B	1B	1B	1B	1B
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	1B	NA	1B	1B
SRAM2SRAM	NA	1B	NA	1B	1B
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	1B	NA	1B	1B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	1B	1B	1B	1B	1B
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	1B	1B	1B	1B	1B
NRAM2WRAM	1B	1B	1B	1B	1B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	NA	NA	NA	NA
WRAM2NRAM	NA	NA	NA	NA	NA
WRAM2WRAM	NA	NA	NA	NA	NA

Instruction Pipeline

- Execute in IO instruction pipeline, if either `<src>` or `<dst>` is off-chip address;
- Execute in Move instruction pipeline, if both `<src>` and `<dst>` are in-chip address.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.3.3 __memcpy_async

```
void __memcpy_async(void *dst,
                   const void *src,
                   int size,
                   mluMemcpyDirection_t dir,
                   int dst_stride,
                   int src_stride,
                   int segnum)
```

Copies `<size>` bytes data from source address `<src>` to destination address `<dst>` asynchronously. The copy direction is specified by `<dir>`.

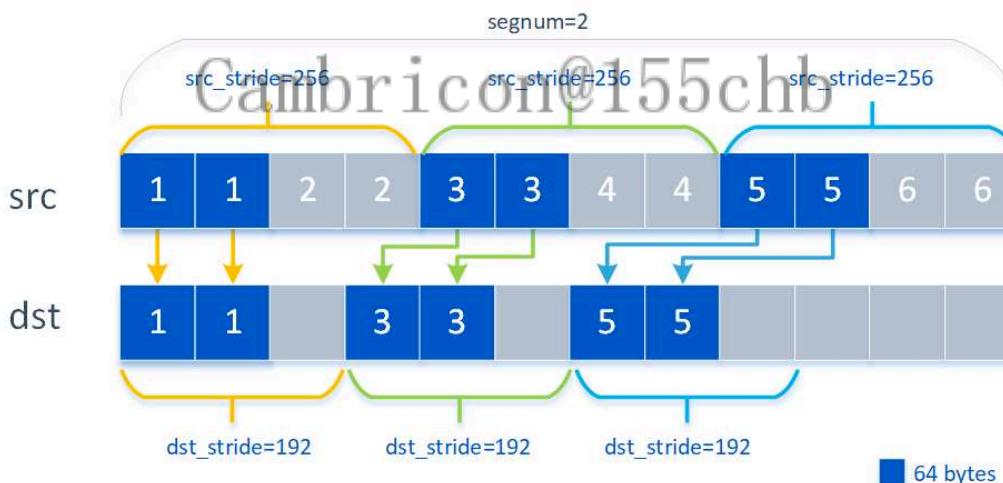


Fig. 3.2: Asynchronous Memory Copy Function with Stride

As shown in Figure [Asynchronous Memory Copy Function with Stride](#), each cell represents 64 bytes and the cells with blue background indicate the data to be copied which is 128 bytes in each segment. There are 3 segments in source area and in each segment only the first 128 bytes of every 256 bytes are copied to destination area. Please note that `<segnum>` is the real segment number minus one, that is, `<segnum>` = 2 in this case.

Parameters

- [out] `dst`: The address of destination area.
- [in] `src`: The address of source area.

- [in] `size`: The number of bytes to be copied.
- [in] `dir`: Copy direction.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnun`: The number of segments minus one.

Return

- `void`.

Remark

- <size> must be greater than zero;
- The range of <segnun> is [0, 65535];
- When the segment number is 0, it means that it is copied once. The real segment number value is the given value plus one;
- <dst_stride> must be greater than or equal to <size>;
- `__memcpy_async` with <id_dst_cluster> is not supported on (m)tp_5xx or higher;
- If the size of the vector with the address space of WRAM is not 8-byte aligned on each LT, the value of the unaligned part is uncertain on tp_322 and (m)tp_372, and is 0 on other targets;
- The alignment constraints of address of <dst> or <src> in `__memcpy_async` are shown in the table [Alignment Constraints of Address Space in `__memcpy`](#);
- The supported copy directions of `__memcpy_async` on different targets are shown in the table [Direction Constraints of `__memcpy_async`](#);

Table 3.11: Direction Constraints of `__memcpy_async`

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2GDRAM	Y	Y	Y	Y	Y
GDRAM2LDRAM	Y	Y	Y	Y	Y
GDRAM2SRAM	N	Y	N	Y	Y
GDRAM2NRAM	Y	Y	Y	Y	Y
GDRAM2WRAM	Y	Y	Y	Y	Y
LDRAM2GDRAM	N	N	N	N	N
LDRAM2LDRAM	N	N	N	N	N
LDRAM2SRAM	N	N	N	N	N
LDRAM2NRAM	N	N	N	N	N
LDRAM2WRAM	N	N	N	N	N
SRAM2GDRAM	N	Y	N	Y	Y
SRAM2LDRAM	N	N	N	N	N

continues on next page

Table 3.11 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
SRAM2SRAM	N	Y	N	Y	Y
SRAM2NRAM	N	Y	N	Y	Y
SRAM2WRAM	N	Y	N	Y	Y
NRAM2GDRAM	Y	Y	Y	Y	Y
NRAM2LDRAM	N	N	N	N	N
NRAM2SRAM	N	Y	N	Y	Y
NRAM2NRAM	Y	Y	Y	Y	Y
NRAM2WRAM	Y	Y	Y	Y	Y
WRAM2GDRAM	N	N	N	N	N
WRAM2LDRAM	N	N	N	N	N
WRAM2SRAM	N	N	N	N	N
WRAM2NRAM	N	N	N	N	N
WRAM2WRAM	N	N	N	N	N

- The alignment constraints of <size> in __memcpy_async are shown in the table Alignment Constraints of size in __memcpy_async;

Table 3.12: Alignment Constraints of size in __memcpy_async

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2GDRAM	NA	NA	NA	NA	NA
GDRAM2LDRAM	NA	NA	NA	NA	NA
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	1B	1B	1B	1B	1B
LDRAM2GDRAM	NA	NA	NA	NA	NA
LDRAM2LDRAM	NA	NA	NA	NA	NA
LDRAM2SRAM	NA	NA	NA	NA	NA

continues on next page

Table 3.12 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
LDRAM2NRAM	NA	NA	NA	NA	NA
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	NA	NA	NA	NA
SRAM2SRAM	NA	NA	NA	NA	NA
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	1B	NA	1B	1B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	NA	NA	NA	NA	NA
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	1B	1B	1B	1B	1B
NRAM2WRAM	1B	1B	1B	1B	1B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	NA	NA	NA	NA
WRAM2NRAM	NA	NA	NA	NA	NA
WRAM2WRAM	NA	NA	NA	NA	NA

- The alignment constraints of <dst_stride> in __memcpy_async are shown in the table [Alignment Constraints of dst_stride in __memcpy_async](#);

Table 3.13: Alignment Constraints of dst_stride in __memcpy_async

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2GDRAM	NA	NA	NA	NA	NA
GDRAM2LDRAM	NA	NA	NA	NA	NA
GDRAM2SRAM	NA	1B	NA	1B	1B

continues on next page

Table 3.13 – continued from previous page

Direction	tp_220 tp_270	mtp_220 mtp_270 mtp_290	tp_322	mtp_322 mtp_372	mtp_592
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	8B	8B	8B	8B	8B
LDRAM2GDRAM	NA	NA	NA	NA	NA
LDRAM2LDRAM	NA	NA	NA	NA	NA
LDRAM2SRAM	NA	NA	NA	NA	NA
LDRAM2NRAM	NA	NA	NA	NA	NA
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	NA	NA	NA	NA
SRAM2SRAM	NA	NA	NA	NA	NA
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	8B	NA	8B	8B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	NA	NA	NA	NA	NA
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	1B	1B	1B	1B	1B
NRAM2WRAM	8B	8B	8B	8B	8B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	NA	NA	NA	NA
WRAM2NRAM	NA	NA	NA	NA	NA
WRAM2WRAM	NA	NA	NA	NA	NA

- The alignment constraints of <src_stride> in __memcpy_async are shown in the table Alignment Constraints of src_stride in __memcpy_async;

Table 3.14: Alignment Constraints of `src_stride` in
`--memcpy_async`

Direction	<code>tp_220</code> <code>tp_270</code>	<code>mtp_220</code> <code>mtp_270</code> <code>mtp_290</code>	<code>tp_322</code>	<code>mtp_322</code> <code>mtp_372</code>	<code>mtp_592</code>
GDRAM2GDRAM	NA	NA	NA	NA	NA
GDRAM2LDRAM	NA	NA	NA	NA	NA
GDRAM2SRAM	NA	1B	NA	1B	1B
GDRAM2NRAM	1B	1B	1B	1B	1B
GDRAM2WRAM	1B	1B	1B	1B	1B
LDRAM2GDRAM	NA	NA	NA	NA	NA
LDRAM2LDRAM	NA	NA	NA	NA	NA
LDRAM2SRAM	NA	NA	NA	NA	NA
LDRAM2NRAM	NA	NA	NA	NA	NA
LDRAM2WRAM	NA	NA	NA	NA	NA
SRAM2GDRAM	NA	1B	NA	1B	1B
SRAM2LDRAM	NA	NA	NA	NA	NA
SRAM2SRAM	NA	NA	NA	NA	NA
SRAM2NRAM	NA	1B	NA	1B	1B
SRAM2WRAM	NA	1B	NA	1B	1B
NRAM2GDRAM	1B	1B	1B	1B	1B
NRAM2LDRAM	NA	NA	NA	NA	NA
NRAM2SRAM	NA	1B	NA	1B	1B
NRAM2NRAM	1B	1B	1B	1B	1B
NRAM2WRAM	1B	1B	1B	1B	1B
WRAM2GDRAM	NA	NA	NA	NA	NA
WRAM2LDRAM	NA	NA	NA	NA	NA
WRAM2SRAM	NA	NA	NA	NA	NA
WRAM2NRAM	NA	NA	NA	NA	NA
WRAM2WRAM	NA	NA	NA	NA	NA

Instruction Pipeline

- Execute in IO instruction pipeline, if either `<src>` or `<dst>` is off-chip address;

- Execute in Move instruction pipeline, if both <src> and <dst> are in-chip address.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.4 3D Memcpy Functions

3.4.1 __bang_move

```
void __bang_move(void *dst,
                 const void *src,
                 int size,
                 int dst_stride0,
                 int dst_segnr0,
                 int dst_stride1,
                 int dst_segnr1,
                 int src_stride0,
                 int src_segnr0,
                 int src_stride1,
                 int src_segnr1)
```

Copies data from <src> to <dst> in 3 dimensions.

As shown in Figure 3D Memory Copy Function with Stride , the cells with blue background indicate <size> of the data to be copied in each segment. In this case, there are 3 segments in the first dimension. In each segment, <src_stride0> of data are copied to destination area <src_segnr0> times. There are 2 segments in the second dimension. In each segment, <src_stride1> of data are copied to destination area <src_segnr1> times. Then, copy the data into corresponding segments and dimensions of <dst>. Please note that <src_segnr0>, <dst_segnr0>, <src_segnr1> and <dst_segnr1> are the real segment number minus one.

Parameters

- [out] dst: The address of destination area.
- [in] src: The address of source area.
- [in] size: The number of bytes of one segment.
- [in] dst_stride0: The destination address stride(bytes) in the first dimension.
- [in] dst_segnr0: The destination segment number in the first dimension.
- [in] dst_stride1: The destination address stride(bytes) in the second dimension.
- [in] dst_segnr1: The destination segment number in the second dimension.
- [in] src_stride0: The source address stride(bytes) in the first dimension.
- [in] src_segnr0: The source segment number in the first dimension.
- [in] src_stride1: The source address stride(bytes) in the second dimension.
- [in] src_segnr1: The source segment number in the second dimension.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero;
- <dst_stride0> and <dst_stride1> must be greater than or equal to <size>;
- <src_stride0> and <src_stride1> must be greater than or equal to 0;
- The total number of iterations at the source must be equal to the total number of iterations at the destination, i.e., ($<\text{dst_segnr}_1 + 1\rangle * <\text{dst_segnr}_2 + 1\rangle = <\text{src_segnr}_1 + 1\rangle * <\text{src_segnr}_2 + 1\rangle$);
- When the segment number of destination and source is 0, it means that it is copied once in this dimension. The real segment number value is the given value plus one;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

```
#include <bang.h>
Cambricon@155chb

#define LEN 7
#define IN_STRIDE_0 11
#define IN_STRIDE_1 103
#define IN_SEGNR_1 7
#define IN_SEGNR_2 2
#define OUT_STRIDE_0 9
#define OUT_STRIDE_1 67
#define OUT_SEGNR_1 5
#define OUT_SEGNR_2 3
#define SIZE (LEN * (IN_SEGNR_1 + 1) * (IN_SEGNR_2 + 1))
#define SIZE_IN (IN_STRIDE_1 * (IN_SEGNR_2 + 1))
#define SIZE_OUT (OUT_STRIDE_1 * (OUT_SEGNR_2 + 1))

__mlu_entry__ void kernel(int8_t *dst, int8_t *src) {
    __nram__ int8_t src1_nram[512];
    __nram__ int8_t src2_nram[512];
    __memcpy(src1_nram, src, 512, GDRAM2NRAM);
    __bang_move(src2_nram, src1_nram, LEN, OUT_STRIDE_0, OUT_SEGNR_1,
                OUT_STRIDE_1, OUT_SEGNR_2, IN_STRIDE_0, IN_SEGNR_1,
                IN_STRIDE_1, IN_SEGNR_2);
    __memcpy(dst, src2_nram, 512, NRAM2GDRAM);
}
```

3.4.2 __memcpy

```
void __memcpy(void *dst,
              const void *src,
              int size,
              mluMemcpyDirection_t dir,
              int dst_stride0,
              int dst_segnr1,
              int dst_stride1,
              int dst_segnr2,
              int src_stride0,
              int src_segnr1,
              int src_stride1,
              int src_segnr2,
              int id_dst_cluster)
```

```
void __memcpy(void *dst,
              const void *src,
              int size,
              mluMemcpyDirection_t dir,
              int dst_stride0,
              int dst_segnr1,
              int dst_stride1,
              int dst_segnr2,
              int src_stride0,
              int src_segnr1,
              int src_stride1,
              int src_segnr2)
```

Copies data from the <src> address space to the <dst> address space in three dimensions. Data operation in the source space is: take the <size> of the data, operate the <src_segnr1> times through the <src_stride0>, and then take the <src_stride1> size to iterate the number of <src_segnr2>. Data operation in the destination space is: take the <size> of the data, operate the <dst_segnr1> times through the <dst_stride0>, and then take the <dst_stride1> size to iterate the number of <dst_segnr2>.

As shown in Figure 3D Memory Copy Function with Stride , the cells with blue background indicate the data to be copied.

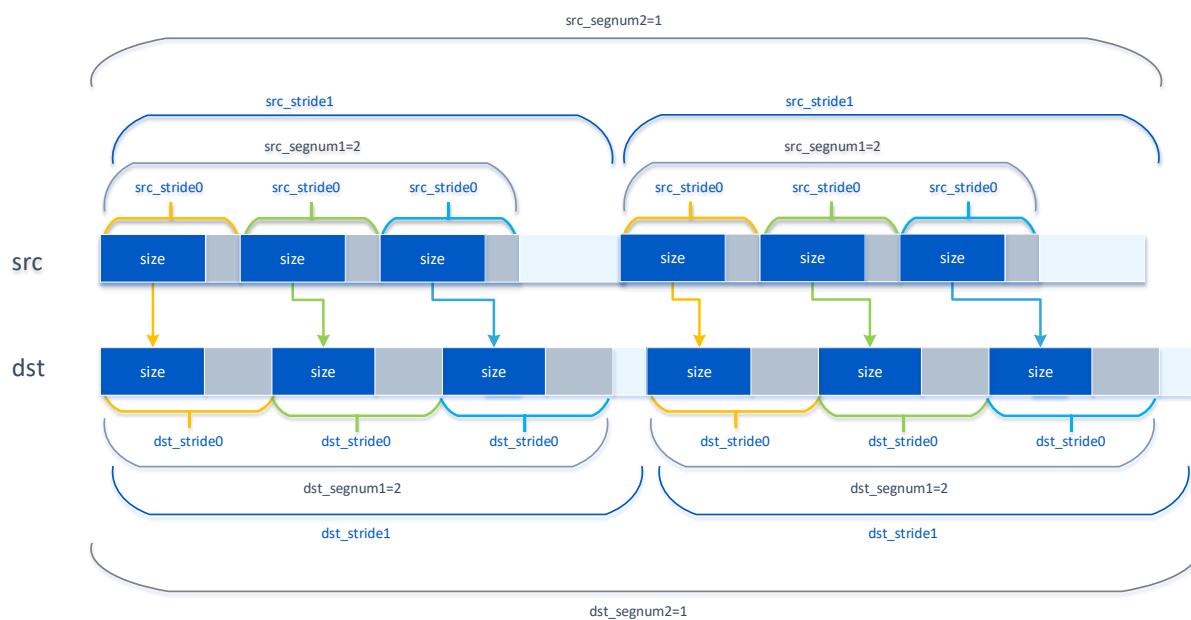


Fig. 3.3: 3D Memory Copy Function with Stride

Parameters

- [out] dst: The address of destination area.
 - [in] src: The address of source area.
 - [in] size: The number of bytes of one segment.
 - [in] dir: Copy direction.
 - [in] dst_stride0: The destination stride(bytes) in the first dimension.
 - [in] dst_segnun1: The destination segment number minus one in the first dimension.
 - [in] dst_stride1: The destination stride(bytes) in the second dimension.
 - [in] dst_segnun2: The destination segment number minus one in the second dimension.
 - [in] src_stride0: The source stride(bytes) in the first dimension.
 - [in] src_segnun1: The source segment number minus one in the first dimension.
 - [in] src_stride1: The source stride(bytes) in the second dimension.
 - [in] src_segnun2: The source segment number minus one in the second dimension.
 - [in] id dst cluster: Destination cluster ID.

Return

- void.

Remark

- When the segment number of destination and source is 0, it means that it is copied once in this dimension. The real segment number value is the given value plus one;
 - The total number of iterations at the source must be equal to the total number of iterations at the destination. That is, the segment numbers need to satisfy:

$$(\text{dst_segnum1} + 1) * (\text{dst_segnum2} + 1) = (\text{src_segnum1} + 1) * (\text{src_segnum2} + 1);$$
 - The `<size>` must be greater than zero, and the absolute value of `<dst_stride0>` must be greater than or equal to `<size>` unless the value of `<dst_stride0>` and `<dst_segnum1>` are equal to zero. The absolute value of `<dst_stride1>` must be greater than or equal

- to <size> unless the value of <dst_stride1> and <dst_segnun2> are equal to zero;
- <id_dst_cluster> is necessarily used when <dir> is SRAM2SRAM and data is copied across different clusters on mtp_2xx and mtp_3xx;
- When <dir> is SRAM2SRAM, copy data across different clusters is not supported on (m)tp_5xx or higher;
- When <dir> is SRAM2SRAM, and there is no <id_dst_cluster>, it means copy data within cluster;
- This function supports transferring data within NRAM, WRAM, SRAM, LDRAM and GDRAM except for WRAM2WRAM;
- When <dst> address space is WRAM, <dst_stride0> and <dst_stride1> must be divisible by 8;
- If the size of the vector with the address space of WRAM is not 8-byte aligned on each LT, the value of the unaligned part is uncertain on tp_322 and (m)tp_372, and is 0 on other targets.

Instruction Pipeline

- Execute in IO instruction pipeline, if either <src> or <dst> is off-chip address;
- Execute in Move instruction pipeline, if both <src> and <dst> are in-chip address.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 7
#define IN_STRIDE_0 11
#define IN_STRIDE_1 103
#define IN_SEGNUM_1 7
#define IN_SEGNUM_2 2
#define OUT_STRIDE_0 9
#define OUT_STRIDE_1 67
#define OUT_SEGNUM_1 5
#define OUT_SEGNUM_2 3

#define SIZE (LEN * (IN_SEGNUM_1 + 1) * (IN_SEGNUM_2 + 1))
#define SIZE_IN (IN_STRIDE_1 * (IN_SEGNUM_2 + 1))
#define SIZE_OUT (OUT_STRIDE_1 * (OUT_SEGNUM_2 + 1))

__mlu_entry__ void kernel(unsigned char *dst, unsigned char *src) {
    __memcpy(dst, src, LEN, GDRAM2GDRAM, OUT_STRIDE_0, OUT_SEGNUM_1,
              OUT_STRIDE_1, OUT_SEGNUM_2, IN_STRIDE_0, IN_SEGNUM_1,
              IN_STRIDE_1, IN_SEGNUM_2);
}
```

3.4.3 __memcpy_async

```
void __memcpy_async(void *dst,
                    const void *src,
                    int size,
                    mluMemcpyDirection_t dir,
                    int dst_stride0,
                    int dst_segnr1,
                    int dst_stride1,
                    int dst_segnr2,
                    int src_stride0,
                    int src_segnr1,
                    int src_stride1,
                    int src_segnr2,
                    int id_dst_cluster)
```

```
void __memcpy_async(void *dst,
                    const void *src,
                    int size,
                    mluMemcpyDirection_t dir,
                    int dst_stride0,
                    int dst_segnr1,
                    int dst_stride1,
                    int dst_segnr2,
                    int src_stride0,
                    int src_segnr1,
                    int src_stride1,
                    int src_segnr2)
```

Cambricon@155chb

Copies data from the <src> address space to the <dst> address space in three dimensions asynchronously. Data operation in the source space is: take the <size> of the data, operate the <src_segnr1> times through the <src_stride0>, and then take the <src_stride1> size to iterate the number of <src_segnr2>. Data operation in the destination space is: take the <size> of the data, operate the <dst_segnr1> times through the <dst_stride0>, and then take the <dst_stride1> size to iterate the number of <dst_segnr2>.

Parameters

- [out] dst: The address of destination area.
- [in] src: The address of source area.
- [in] size: The number of bytes of one segment.
- [in] dir: Copy direction.
- [in] dst_stride0: The destination stride(bytes) in the first dimension.
- [in] dst_segnr1: The destination segment number minus one in the first dimension.
- [in] dst_stride1: The destination stride(bytes) in the second dimension.
- [in] dst_segnr2: The destination segment number minus one in the second dimension.
- [in] src_stride0: The source stride(bytes) in the first dimension.

- [in] `src_segnun1`: The source segment number minus one in the first dimension.
- [in] `src_stride1`: The source stride(bytes) in the second dimension.
- [in] `src_segnun2`: The source segment number minus one in the second dimension.
- [in] `id_dst_cluster`: Destination cluster ID.

Return

- void.

Remark

- When the segment number of destination and source is 0, it means that it is copied once in this dimension. The real segment number value is the given value plus one;
- The total number of iterations at the source must be equal to the total number of iterations at the destination. That is, the segment numbers need to satisfy: $(\text{dst_segnun1} + 1) * (\text{dst_segnun2} + 1) = (\text{src_segnun1} + 1) * (\text{src_segnun2} + 1)$;
- The `<size>` must be greater than zero, and the absolute value of `<dst_stride0>` must be greater than or equal to `<size>` unless the value of `<dst_stride0>` and `<dst_segnun1>` are equal to zero. The absolute value of `<dst_stride1>` must be greater than or equal to `<size>` unless the value of `<dst_stride1>` and `<dst_segnun2>` are equal to zero;
- `<id_dst_cluster>` is necessarily used when `<dir>` is SRAM2SRAM and data is copied across different clusters on `mtp_2xx`, `mtp_3xx`;
- When `<dir>` is SRAM2SRAM, copy data across different clusters is not supported on `(m)tp_5xx` or higher;
- When `<dir>` is SRAM2SRAM, and there is no `<id_dst_cluster>`, it means copy data within cluster;
- This function supports transferring data within NRAM, WRAM, SRAM, LDRAM and GDRAM except for WRAM2WRAM;
- When `<dst>` address space is WRAM, `<dst_stride0>` and `<dst_stride1>` must be divisible by 8;
- If the size of the vector with the address space of WRAM is not 8-byte aligned on each LT, the value of the unaligned part is uncertain on `tp_322` and `(m)tp_372`, and is 0 on other targets.

Instruction Pipeline

- Execute in IO instruction pipeline, if either `<src>` or `<dst>` is off-chip address;
- Execute in Move instruction pipeline, if both `<src>` and `<dst>` are in-chip address.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.2.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define LEN 7
#define IN_STRIDE_0 11
#define IN_STRIDE_1 103
#define IN_SEGNUM_1 7
#define IN_SEGNUM_2 2
#define OUT_STRIDE_0 9
```

```
#define OUT_STRIDE_1 67
#define OUT_SEGNUM_1 5
#define OUT_SEGNUM_2 3
#define SIZE (LEN * (IN_SEGNUM_1 + 1) * (IN_SEGNUM_2 + 1))
#define SIZE_IN (IN_STRIDE_1 * (IN_SEGNUM_2 + 1))
#define SIZE_OUT (OUT_STRIDE_1 * (OUT_SEGNUM_2 + 1))

__mlu_entry__ void kernel(unsigned char *dst, unsigned char *src) {
    __memcpy_async(dst, src, LEN, GDRAM2GDRAM, OUT_STRIDE_0, OUT_SEGNUM_1,
                   OUT_STRIDE_1, OUT_SEGNUM_2, IN_STRIDE_0, IN_SEGNUM_1,
                   IN_STRIDE_1, IN_SEGNUM_2);
}
```

3.5 3D Memset Functions

3.5.1 __bang_write_value

```
void __bang_write_value(void *dst,
                        int elem_count,
                        char value,
                        int stride0,
                        int iter1,
                        int stride1,
                        int iter2)
```

```
void __bang_write_value(void *dst,
                        int elem_count,
                        half value,
                        int stride0,
                        int iter1,
                        int stride1,
                        int iter2)
```

```
void __bang_write_value(void *dst,
                        int elem_count,
                        bfloat16_t value,
                        int stride0,
                        int iter1,
                        int stride1,
                        int iter2)
```

```
void __bang_write_value(void *dst,
                        int elem_count,
                        short value,
                        int stride0,
                        int iter1,
                        int stride1,
                        int iter2)
```

```
void __bang_write_value(void *dst,
                        int elem_count,
                        float value,
                        int stride0,
                        int iter1,
                        int stride1,
                        int iter2)
```

```
void __bang_write_value(void *dst,
                        int elem_count,
                        int value,
                        int stride0,
                        int iter1,
                        int stride1,
                        int iter2)
```

Cambricon@155chb

Sets a three-dimensional block in the `__nram__` address space pointed by `<dst>` to the specified `<value>`. The way of setting `<value>` is that takes the `<elem_count>` of the data, operates the `<iter1> + 1` times through the `<stride0>`, and then take the `<stride1>` size to iterate the number of `<iter2> + 1`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `elem_count`: The number of elements of one segment.
- [in] `value`: The value to be set to `<dst>`.
- [in] `stride0`: The source address stride(bytes) in the first dimension.
- [in] `iter1`: The source segment number in the first dimension.
- [in] `stride1`: The source address stride(bytes) in the second dimension.
- [in] `iter2`: The source segment number in the second dimension.

Return

- `void`.

Remark

- `<dst>` must point to `__nram__` address space;
- `<elem_count>` must be greater than zero;
- `<stride0>` must be greater than or equal to `<elem_count> * sizeof(<value>)`, and `<stride1>` must be greater than or equal to `<stride0> * (<iter1> + 1)`;
- `<iter1>` and `<iter2>` must be greater than or equal to zero; When the value of `<iter1>` and `<iter2>` is 0, it represents 1, and when the value is 1, it represents 2, and so on;
- `bfloat16_t` is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.2.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx .`

Example

- None.

3.5.2 __gdramset

```
void __gdramset(void *dst,
                 int elem_count,
                 char value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 half value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 bfloat16_t value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 short value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 float value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __gdramset(void *dst,
                 int elem_count,
                 int value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

Sets the specified <value> to a three-dimensional block in the `__mlu_device__` address space pointed by <dst>. The way of setting <value> is that takes the <elem_count> of the data, operates the <iter1> + 1 times through the <stride0>, and then takes the <stride1> size to iterate the number of <iter2> + 1.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `elem_count`: The number of elements of one segment.
- [in] `value`: The value to be set to <dst>.
- [in] `stride0`: The source address stride(bytes) in the first dimension.
- [in] `iter1`: The source segment number in the first dimension.
- [in] `stride1`: The source address stride(bytes) in the second dimension.
- [in] `iter2`: The source segment number in the second dimension.

Return

- `void`.

Remark

- <dst> must point to `__mlu_device__` address space;
- <elem_count> must be greater than zero;
- <stride0> must be greater than or equal to <elem_count> * `sizeof(<value>)`, and <stride1> must be greater than or equal to <stride0> * (<iter1> + 1);
- <iter1> and <iter2> must be greater than or equal to zero; When <iter1> and <iter2> are 0, they represent 1, and when <iter1> and <iter2> are 1, they represent 2, and so on;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

- None.

3.5.3 __lDRAMset

```
void __lDRAMset(void *dst,
                 int elem_count,
                 char value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 half value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 bfloat16_t value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 short value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __lDRAMset(void *dst,
                 int elem_count,
                 float value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

```
void __1dramset(void *dst,
                 int elem_count,
                 int value,
                 int stride0,
                 int iter1,
                 int stride1,
                 int iter2)
```

Sets the specified <value> to a three-dimensional block in the `__1dram__` address space pointed by <dst>. The way of setting <value> is that takes the <elem_count> of the data, operates the <iter1> + 1 times through the <stride0>, and then takes the <stride1> size to iterate the number of <iter2> + 1.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements of one segment.
- [in] value: The value to be set to <dst>.
- [in] stride0: The source address stride(bytes) in the first dimension.
- [in] iter1: The source segment number in the first dimension.
- [in] stride1: The source address stride(bytes) in the second dimension.
- [in] iter2: The source segment number in the second dimension.

Return

- void.

Remark

Cambricon@155chb

- <dst> must point to `__1dram__` address space;
- <elem_count> must be greater than zero;
- <stride0> must be greater than or equal to <elem_count> * sizeof(<value>), and <stride1> must be greater than or equal to <stride0> * (<iter1> + 1);
- <iter1> and <iter2> must be greater than or equal to zero; When <iter1> and <iter2> are 0, they represent 1, and when <iter1> and <iter2> are 1, they represent 2, and so on;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

- None.

3.5.4 __nramset

```
void __nramset(void *dst,
               int elem_count,
               char value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __nramset(void *dst,
               int elem_count,
               half value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __nramset(void *dst,
               int elem_count,
               bfloat16_t value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __nramset(void *dst,
               int elem_count,
               short value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __nramset(void *dst,
               int elem_count,
               float value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __nramset(void *dst,
               int elem_count,
               int value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

Sets the specified <value> to a three-dimensional block in the __nram__ address space pointed by <dst>. The way of setting <value> is that takes the <elem_count> of the data, operates the <iter1> + 1 times through the <stride0>, and then takes the <stride1> size to iterate the number of <iter2> + 1.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements of one segment.
- [in] value: The value to be set to <dst>.
- [in] stride0: The source address stride(bytes) in the first dimension.
- [in] iter1: The source segment number in the first dimension.
- [in] stride1: The source address stride(bytes) in the second dimension.
- [in] iter2: The source segment number in the second dimension.

Return

- void.

Remark

Cambricon@155chb

- <dst> must point to __nram__ address space;
- <elem_count> must be greater than zero;
- <stride0> must be greater than or equal to <elem_count> * sizeof(<value>), and <stride1> must be greater than or equal to <stride0> * (<iter1> + 1);
- <iter1> and <iter2> must be greater than or equal to zero; When <iter1> and <iter2> are 0, they represent 1, and when <iter1> and <iter2> are 1, they represent 2, and so on;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Move.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.5.5 __sramset

```
void __sramset(void *dst,
               int elem_count,
               char value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __sramset(void *dst,
               int elem_count,
               half value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __sramset(void *dst,
               int elem_count,
               bfloat16_t value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __sramset(void *dst,
               int elem_count,
               short value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __sramset(void *dst,
               int elem_count,
               float value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

```
void __sramset(void *dst,
               int elem_count,
               int value,
               int stride0,
               int iter1,
               int stride1,
               int iter2)
```

Sets the specified <value> to a three-dimensional block in the `__mlu_shared__` address space pointed by <dst>. The way of setting <value> is that takes the <elem_count> of the data, operates the <iter1> + 1 times through the <stride0>, and then takes the <stride1> size to iterate the number of <iter2> + 1.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements of one segment.
- [in] value: The value to be set to <dst>.
- [in] stride0: The source address stride(bytes) in the first dimension.
- [in] iter1: The source segment number in the first dimension.
- [in] stride1: The source address stride(bytes) in the second dimension.
- [in] iter2: The source segment number in the second dimension.

Return

- void.

Remark

Cambricon@155chb

- <dst> must point to `__mlu_shared__` address space;
- <elem_count> must be greater than zero;
- <stride0> must be greater than or equal to <elem_count> * sizeof(<value>), and <stride1> must be greater than or equal to <stride0> * (<iter1> + 1);
- <iter1> and <iter2> must be greater than or equal to zero; When <iter1> and <iter2> are 0, they represent 1, and when <iter1> and <iter2> are 1, they represent 2, and so on;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Move.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

- None.

3.6 Artificial Intelligence Functions

3.6.1 __bang_avgpool

```
void __bang_avgpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width)
```

```
void __bang_avgpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_avgpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_avgpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_avgpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_avgpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_avgpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

Performs avgpooling forward propagation operation on <src> [`<height>, <width>, <channel>`], a three-dimensional tensor, with sliding window [`<kernel_height>, <kernel_width>`] and stride [`<stride_width>, <stride_height>`], and calculates the average value in each window. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of input feature map.
- [in] width: The width of input feature map.
- [in] kernel_height: The height of kernel.
- [in] kernel_width: The width of kernel.
- [in] stride_width: Stride of sliding window in W direction.
- [in] stride_height: Stride of sliding window in H direction.
- [in] in_dh: Dilation in H direction of input.
- [in] in_dw: Dilation in W direction of input.
- [in] out_dh: Dilation in H direction of output.
- [in] out_dw: Dilation in W direction of output.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <kernel_height> and <kernel_width> must be greater than or equal to 1;
- <stride_height> and <stride_width> must be greater than or equal to 1 if specified;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The function with dilation is supported on (m)tp_3xx or higher;
- <dst> cannot be overlapped with <src>;
- In the function without stride, <stride_width> equals <kernel_width> and <stride_height> equals <kernel_height>;
- (*kernel_height* × *kernel_width*) < 2¹⁶;
- <channel>, <in_dh>, <in_dw>, <out_dh> and <out_dw> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define CHANNELS 64
#define HEIGHT 9
#define WIDTH 9
#define KERNEL_HEIGHT 3
#define KERNEL_WIDTH 3
#define BOTTOM_DATA_COUNT ((CHANNELS) * (WIDTH) * (HEIGHT))
#define TOP_DATA_COUNT \
    ((CHANNELS) * (HEIGHT / KERNEL_HEIGHT) * (WIDTH / KERNEL_WIDTH))
```

```
--mlu_entry__ void avgPoolingKernel(half* bottom_data, half* top_data,
                                    int channels, int height, int width,
                                    int pooled_height, int pooled_width) {
    __nram__ half a_tmp[BOTTOM_DATA_COUNT];
    __nram__ half b_tmp[TOP_DATA_COUNT];
    __memcpy(a_tmp, bottom_data, BOTTOM_DATA_COUNT * sizeof(half), GDRAM2NRAM);
    __bang_avgpool(b_tmp, a_tmp, CHANNELS, HEIGHT, WIDTH, KERNEL_HEIGHT, KERNEL_WIDTH);
    __memcpy(top_data, b_tmp, TOP_DATA_COUNT * sizeof(half), NRAM2GDRAM);
}
```

3.6.2 __bang_avgpool_bp

```
void __bang_avgpool_bp(half*dst,
                      half*src,
                      int channel,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      mluPoolBPOverlap overlap = OVERLAP_ACC)
```

```
void __bang_avgpool_bp(bfloat16_t *dst,
                      bfloat16_t *src,
                      int channel,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      mluPoolBPOverlap overlap = OVERLAP_ACC)
```

```
void __bang_avgpool_bp(float *dst,
                      float *src,
                      int channel,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      mluPoolBPOverlap overlap = OVERLAP_ACC)
```

Performs avgpooling backward propagation operation on <src> [, ,

`<channel>`], a three-dimensional tensor, with sliding window [`<kernel_height>`, `<kernel_width>`] and stride [`<stride_width>`, `<stride_height>`], and calculates the average value in each window. `<overlap>` indicates the type of overlap options. `<overlap>` is assigned to an enumerated type called `mluPoolBPOverlap` that contains 2 enumerators listed in the table below. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded.

Table 3.15: Semantics of `mluPoolBPOverlap`

<code>mluPoolBPOverlap</code> Type	Semantic
<code>OVERLAP_ACC</code>	Accumulates the overlap parts of the output.
<code>OVERLAP_COVER</code>	Covers the overlap parts of the output.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination tensor whose data layout is HWC.
- [in] `src`: The address of source tensor whose data layout is HWC.
- [in] `channel`: Input channel.
- [in] `height`: The height of output feature map.
- [in] `width`: The width of output feature map.
- [in] `kernel_height`: The height of kernel.
- [in] `kernel_width`: The width of kernel.
- [in] `stride_width`: Stride of sliding window in W direction.
- [in] `stride_height`: Stride of sliding window in H direction.
- [in] `overlap`: The type of overlap options.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<channel> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<dst>` cannot be overlapped with `<src>`;
- The default `mluPoolBPOverlap` option is `OVERLAP_ACC`;
- `<channel>`, `<stride_height>` and `<stride_width>` must be greater than 0;
- `<kernel_height>` and `<kernel_width>` must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

__mlu_entry__ void PoolAvgBpKernel(half* output, half* input,
                                    int channels, int out_height,
                                    int out_width, int kernel_height,
                                    int kernel_width, int stride_width,
                                    int stride_height) {
    __nram__ half a_tmp[INPUT_COUNT];
    __nram__ half b_tmp[OUTPUT_COUNT];
    __memcpy(b_tmp, output, OUTPUT_COUNT * sizeof(half), GDRAM2NRAM);
    __memcpy(a_tmp, input, INPUT_COUNT * sizeof(half), GDRAM2NRAM);
    __bang_avgpool_bp(b_tmp, a_tmp, channels, out_height, out_width,
                      kernel_height, kernel_width, stride_width,
                      stride_height);
    __memcpy(output, b_tmp, OUTPUT_COUNT * sizeof(half), NRAM2GDRAM);
}
```

3.6.3 __bang_bexpand

```
void __bang_bexpand(char *dst,
                     unsigned char *src,
                     int dst_dim_n,
                     int dst_dim_h,
                     int dst_dim_w,
                     int byte_of_dst_dim_c)
```

```
void __bang_bexpand(short *dst,
                     unsigned char *src,
                     int dst_dim_n,
                     int dst_dim_h,
                     int dst_dim_w,
                     int byte_of_dst_dim_c)
```

```
void __bang_bexpand(int *dst,
                     unsigned char *src,
                     int dst_dim_n,
                     int dst_dim_h,
                     int dst_dim_w,
                     int byte_of_dst_dim_c)
```

This function converts each binary bit in `<src>` to value of specified data type and saves the result in `<dst>`. The bit 1 and 0 will be converted to value 1 and 0 of corresponding data type. For example, if the data type of `<dst>` is `char` and `<src>` is `0b11001101`, the `<dst>` will be `0b 00000001 00000001 00000000 00000000 00000001 00000001 00000000 00000001`.

Parameters

- [out] `dst`: The address of destination tensor.

- [in] src: The address of source binary vector.
- [in] dst_dim_n: The dimension N of destination tensor.
- [in] dst_dim_h: The dimension H of destination tensor.
- [in] dst_dim_w: The dimension W of destination tensor.
- [in] byte_of_dst_dim_c: The byte of dimension C of destination tensor.

Return

- void.*

Remark

- <dst> and <src> must point to __nram__ address space;
- <dst> cannot be overlapped with <src>;
- <dst_dim_n>, <dst_dim_h> and <dst_dim_w> are integers and all of them must be greater than or equal to 1;
- The maximum value of <dst_dim_n>, <dst_dim_h> and <dst_dim_w> is 65535;
- <byte_of_dst_dim_c> is equal to dst_dim_c * sizeof(dst_data_type) . For example, if the data type of <dst> is int and the dimension C of destination tensor is 16, the <byte_of_dst_dim_c> is 16 * 4 = 64(Byte) ;
- <byte_of_dst_dim_c> must be greater than zero and divisible by 8 * sizeof(dst_data_type) ;
- The maximum value of <byte_of_dst_dim_c> is 524288.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```
#include "bang.h"
#define N 1
#define H 2
#define W 8
#define C 32
#define DST_NUM (N * H * W * C)
#define SRC_NUM (DST_NUM / 8)

__mlu_entry__ void kernel(int* dst, unsigned char* src) {
    __nram__ unsigned char src_tmp[SRC_NUM];
    __nram__ int dst_tmp[DST_NUM];
    __memcpy(src_tmp, src, SRC_NUM * sizeof(unsigned char), GDRAM2NRAM);
    __bang_bexpand(dst_tmp, src_tmp, N, H, W, C * sizeof(int));
    __memcpy(dst, dst_tmp, DST_NUM * sizeof(int), NRAM2GDRAM);
}
```

3.6.4 __bang_breduce

```
void __bang_breduce(unsigned char *dst,
                    char *src,
                    int src_dim_n,
                    int src_dim_h,
                    int src_dim_w,
                    int byte_of_src_dim_c)
```

```
void __bang_breduce(unsigned char *dst,
                    short *src,
                    int src_dim_n,
                    int src_dim_h,
                    int src_dim_w,
                    int byte_of_src_dim_c)
```

```
void __bang_breduce(unsigned char *dst,
                    int *src,
                    int src_dim_n,
                    int src_dim_h,
                    int src_dim_w,
                    int byte_of_src_dim_c)
```

This function converts each value in <src> to binary bit and saves the result in <dst>. Value 0 will be converted to bit 0 and other value will be converted to bit 1. For example, if the data type of <src> is `char` and <src> is `0b 00000001 10000001 00000000 00000000 11111111 01000000 00000000 11000000`, the <dst> will be `0b11001101`.

Parameters

- [out] `dst`: The address of destination binary vector.
- [in] `src`: The address of source tensor.
- [in] `src_dim_n`: The dimension N of source tensor.
- [in] `src_dim_h`: The dimension H of source tensor.
- [in] `src_dim_w`: The dimension W of source tensor.
- [in] `byte_of_src_dim_c`: The byte of dimension C of source tensor.

Return

- `void *`

Remark

- <dst> and <src> must point to `__nram__` address space;
- <dst> cannot be overlapped with <src>;
- <src_dim_n>, <src_dim_h> and <src_dim_w> are integers and all of them must be greater than or equal to 1;
- The maximum value of <src_dim_n>, <src_dim_h> and <src_dim_w> is 65535;
- <byte_of_src_dim_c> is equal to `src_dim_c * sizeof(src_data_type)`. For example, if the data type of <src> is `int` and the dimension C of source tensor is 16, the <byte_of_src_dim_c> is `16 * 4 = 64(Byte)`;
- <byte_of_src_dim_c> must be greater than zero and divisible by `8 * sizeof(src_data_type)`.

```
    sizeof(src_data_type);  
• The maximum value of <byte_of_src_dim_c> is 524288.
```

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```
#include "bang.h"  
  
#define N 1  
#define H 2  
#define W 8  
#define C 32  
  
#define SRC_NUM (N * H * W * C)  
#define DST_NUM (SRC_NUM / 8)  
  
__mlu_entry__ void kernel(unsigned char* dst, int* src) {  
    __nram__ int src_tmp[SRC_NUM];  
    __nram__ unsigned char dst_tmp[DST_NUM];  
    __memcpy(src_tmp, src, SRC_NUM * sizeof(int), GDRAM2NRAM);  
    __bang_breduce(dst_tmp, src_tmp, N, H, W, C * sizeof(int));  
    __memcpy(dst, dst_tmp, DST_NUM * sizeof(unsigned char), NRAM2GDRAM);  
}
```

Cambricon@155chb

3.6.5 __bang_conv

```
void __bang_conv(float *dst,  
                 int32_t *src,  
                 int16_t *kernel,  
                 const int channel_input,  
                 const int height,  
                 const int width,  
                 const int kernel_height,  
                 const int kernel_width,  
                 const int stride_x,  
                 const int stride_y,  
                 const int channel_output,  
                 int fix_position)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int8_t *kernel,
                 int16_t *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int16_t *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int16_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                 int8_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                  int8_t *src,
                  int8_t *kernel,
                  int16_t *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                  int16_t *src,
                  int8_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int8_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                 int8_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(float *dst,
                 int8_t *src,
                 int8_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int16_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                  int8_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int16_t *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                 int8_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int8_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(int16_t *dst,
                  int8_t *src,
                  int8_t *kernel,
                  int16_t *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(int16_t *dst,
                  int16_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(int16_t *dst,
                 int16_t *src,
                 int8_t *kernel,
                 int16_t *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int8_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int16_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                 int16_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int8_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                  int8_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                  int8_t *src,
                  int8_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int8_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int16_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int16_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  half *src,
                  half *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output)
```

```
void __bang_conv(float *dst,
                  float *src,
                  float *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output)
```

```
void __bang_conv(float *dst,
                 half *src,
                 half *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output)
```

```
void __bang_conv(float *dst,
                 float *src,
                 half *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output)
```

```
void __bang_conv(float *dst,
                 half *src,
                 float *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output)
```

```
void __bang_conv(float *dst,
                 bfloat16_t *src,
                 float *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output)
```

```
void __bang_conv(float *dst,
                 float *src,
                 bfloat16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output)
```

```
void __bang_conv(float *dst,
                 bfloat16_t *src,
                 bfloat16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output)
```

```
void __bang_conv(float *dst,
                 float *src,
                 float *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

```
void __bang_conv(float *dst,
                 half *src,
                 half *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

```
void __bang_conv(float *dst,
                  half *src,
                  float *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(float *dst,
                  float *src,
                  half *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(float *dst,
                 float *src,
                 bfloat16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

```
void __bang_conv(float *dst,
                 bfloat16_t *src,
                 float *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

```
void __bang_conv(half *dst,
                 int4x2_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(float *dst,
                 int4x2_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                 int4x2_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(float *dst,
                 int4x2_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                 int8_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                 int4x2_t *src,
                 int8_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                 int4x2_t *src,
                 int16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                 int4x2_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int fix_position,
                  int channel_output,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int4x2_t *kernel,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height,
                  int outdilation_width,
                  int outdilation_height)
```

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int4x2_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int4x2_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                 int4x2_t *src,
                 int4x2_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                 int4x2_t *src,
                 int16_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                 int8_t *src,
                 int4x2_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position)
```

```
void __bang_conv(half *dst,
                  int16_t *src,
                  int4x2_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(float *dst,
                  int16_t *src,
                  int4x2_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int4x2_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int4x2_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int8_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(half *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                  int4x2_t *src,
                  int16_t *kernel,
                  float *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

```
void __bang_conv(half *dst,
                  int8_t *src,
                  int4x2_t *kernel,
                  half *bias,
                  int channel_input,
                  int height,
                  int width,
                  int kernel_height,
                  int kernel_width,
                  int stride_width,
                  int stride_height,
                  int channel_output,
                  int fix_position,
                  int indilation_width,
                  int indilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                 int8_t *src,
                 int4x2_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(half *dst,
                 int16_t *src,
                 int4x2_t *kernel,
                 half *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

Cambricon@155chb

```
void __bang_conv(float *dst,
                 int16_t *src,
                 int4x2_t *kernel,
                 float *bias,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int fix_position,
                 int indilation_width,
                 int indilation_height)
```

```
void __bang_conv(float *dst,
                 bfloat16_t *src,
                 bfloat16_t *kernel,
                 int channel_input,
                 int height,
                 int width,
                 int kernel_height,
                 int kernel_width,
                 int stride_width,
                 int stride_height,
                 int channel_output,
                 int indilation_width,
                 int indilation_height,
                 int outdilation_width,
                 int outdilation_height)
```

In `__nram__` address space, uses the four-dimensional convolution kernel `<kernel>[<channel_output>, <kernel_height>, <kernel_width>, <channel_input>]` to perform a convolution operation on the three-dimensional tensor `<src>[<height>, <width>, <channel_input>]` with an interactive step size of `[<stride_width>, <stride_height>]`, and stores the result in the three-dimensional tensor `<dst>[<dst_height>, <dst_width>, <channel_output>]`.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination tensor which has $NH_oW_oC_o$ data layout.
- [in] `src`: The address of source tensor which has $NH_iW_iC_i$ data layout.
- [in] `kernel`: The address of filter tensor which has $C_oH_kW_kC_i$ data layout.
- [in] `bias`: The address of bias tensor which has $[C_o]$ shape.

- [in] channel_input: Number of input channels.
- [in] height: The height of source tensor.
- [in] width: The width of source tensor.
- [in] kernel_height: The height of filter tensor.
- [in] kernel_width: The width of filter tensor.
- [in] stride_width: The stride in W direction.
- [in] stride_height: The stride in H direction.
- [in] channel_output: Number of output channels.
- [in] fix_position: Sum of the scale factor of <src> and <kernel>.
- [in] indilation_width: Input dilation in W direction.
- [in] indilation_height: Input dilation in H direction.
- [in] outdilation_width: Output dilation in W direction.
- [in] outdilation_height: Output dilation in H direction.

Return

- void.

Remark

- <src>, <dst> and <bias> must point to __nram__ address space;
- <kernel> must point to __wram__ address space;
- <fix_position> is the sum of the scale factor of <src> and <kernel>;
- <fix_position> must be in the range [-127, 127];
- The address of <dst>, <src> and <bias> must be 64-byte aligned on (m)tp_2xx;
- The address of <kernel> must be 32-byte aligned;
- <channel_input> * sizeof(typeof<src>) must be 64-byte aligned on (m)tp_2xx;
- <channel_output> must be divisible by 64 on (m)tp_2xx;
- <channel_output> * sizeof(typeof<dst>) must be 64-byte aligned on (m)tp_2xx;
- bang_conv with int input version is not recommended, because it is implemented by combination, with lower performance. bang_device_functions_extra.h should be included when using this version. When <channel_input> * <width> * <height> > 12800, 12800 / (<width> * <height>) should be divisible by 32;
- The byte size of <kernel> must be 64-byte aligned; otherwise, the non-aligned part of <kernel> will be random numbers on mtp_372 and tp_322, and zero on other architecture;
- On (m)tp_3xx and higher, ci dimension space of <kernel> must satisfy the following alignment constraints:
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>) == 4$, $\text{ci} * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 16-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>) == 2$, $\text{ci} * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 32-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>)$ doesn't satisfy the above mentioned constraints, $\text{ci} * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 64-byte aligned;
- co dimension space of <kernel> must be divisible by 64;
- The space size of <bias> is identical to <channel_output>;
- <kernel_height>, <kernel_width>, <channel_input> and <channel_output> must be greater than 0;
- <stride_width> and <stride_height> must be in range [1, 1023];
- <indilation_width>, <indilation_height>, <outdilation_width> and <outdilation_height> must be in range [1, 1023], and 1 means no dilation;

- If `<indilation_height>` is used, $(\text{height} - ((\text{kernel_height} - 1) * \text{indilation_height} + 1)) / \text{stride_height} + 1$ must be greater than 0; otherwise, $(\text{height} - \text{kernel_height}) / \text{stride_height} + 1$ must be greater than 0;
- If `<indilation_width>` is used, $(\text{width} - ((\text{kernel_width} - 1) * \text{indilation_width} + 1)) / \text{stride_width} + 1$ must be greater than 0; otherwise, $(\text{width} - \text{kernel_width}) / \text{stride_width} + 1$ must be greater than 0;
- On target (m)tp_322, `<kernel>` must be `int4x2`, `int8_t` or `int16_t`;
- On target (m)tp_322, if use `<indilation_height>`, $(\text{height} - ((\text{kernel_height} - 1) * \text{indilation_height} + 1)) / \text{stride_height} + 1$ must be in the range [1, 1024] ;
- On target (m)tp_322, if use `<dilation_height>`, $(\text{height} - ((\text{kernel_height} - 1) * \text{dilation_height} + 1)) / \text{stride_height} + 1$ must be in the range [1, 1024] ;
- On target (m)tp_322, if neither `<dilation_height>` nor `<indilation_height>` are used, $(\text{height} - \text{kernel_height}) / \text{stride_height} + 1$ must be in range [1, 1024] ;
- `<dst>` cannot be overlapped with `<src>`;
- `<dst>` cannot be overlapped with `<bias>`.

Compatibility between Various Architectures

Table 3.16: Conv Data Types Supported on (m)tp_2xx

Src Type	Kernel Type	Dst Type	Bias Type
int4	int4	half	none
int4	int4	float	none
int8	int4	half	none
int8	int4	float	none
int8	int8	half	none
int8	int8	float	none
int8	int8	int16	none
int16	int4	half	none
int16	int4	float	none
int16	int8	half	none
int16	int8	float	none
int16	int8	int16	none
int16	int16	half	none
int16	int16	float	none

continues on next page

Table 3.16 – continued from previous page

Src Type	Kernel Type	Dst Type	Bias Type
int16	int16	int16	none
int	int16	float	none
int4	int4	half	half
int4	int4	float	float
int8	int4	half	half
int8	int4	float	float
int8	int8	half	half
int8	int8	float	float
int8	int8	int16	int16
int16	int4	half	half
int16	int4	float	float
int16	int8	half	half
int16	int8	float	float
int16	int8	int16	int16
int16	int16	half	half
int16	int16	float	float
int16	int16	int16	int16

Table 3.17: Conv Data Types Supported on (m)tp_3xx

Src Type	Kernel Type	Dst Type	Bias Type
int8	int8	half	none
int8	int8	float	none
int8	int8	int16	none
int16	int8	half	none
int16	int8	float	none
int16	int8	int16	none
int16	int16	half	none
int16	int16	float	none

continues on next page

Table 3.17 – continued from previous page

Src Type	Kernel Type	Dst Type	Bias Type
int16	int16	int16	none
int	int16	float	none
int8	int8	half	half
int8	int8	float	float
int8	int8	int16	int16
int16	int8	half	half
int16	int8	float	float
int16	int8	int16	int16
int16	int16	half	half
int16	int16	float	float
int16	int16	int16	int16
float	float	float	none
half	half	float	none
float	half	float	none
half	float	float	none
bfloat16_t	bfloat16_t	float	none
float	bfloat16_t	float	none
bfloat16_t	float	float	none
int4	int4	float	none
int4	int8	float	none
int4	int16	float	none
int8	int4	float	none
int16	int4	float	none
int4	int4	half	none
int4	int8	half	none
int4	int16	half	none
int8	int4	half	none

continues on next page

Table 3.17 – continued from previous page

Src Type	Kernel Type	Dst Type	Bias Type
int16	int4	half	none
int4	int4	float	float
int4	int8	float	float
int4	int16	float	float
int8	int4	float	float
int16	int4	float	float
int4	int4	half	half
int4	int8	half	half
int4	int16	half	half
int8	int4	half	half
int16	int4	half	half

Instruction Pipeline

- Compute.

Requirements

- Cambricon@155chb**
- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
 - CNCC Version: `cncc --version >= 2.8.0;`
 - Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
 - MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__bang_conv_partial` for more details.

3.6.6 `__bang_conv_partial`

```
void __bang_conv_partial(half *dst,
                        int8_t *src,
                        int8_t *kernel,
                        half *partial,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int16_t *src,
                         int8_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int16_t *src,
                         int16_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(float *dst,
                         int8_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int8_t *src,
                         int8_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int16_t *src,
                         int8_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int16_t *src,
                         int16_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int8_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(float *dst,
                         half *src,
                         half *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         half *src,
                         float *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         float *src,
                         half *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         float *src,
                         float *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         bfloat16_t *src,
                         bfloat16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         bfloat16_t *src,
                         float *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         float *src,
                         bfloat16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output)
```

```
void __bang_conv_partial(float *dst,
                         half *src,
                         half *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         float *src,
                         float *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         float *src,
                         half *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         half *src,
                         float *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         float *src,
                         bfloat16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         bfloat16_t *src,
                         bfloat16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(half *dst,
                        int4x2_t *src,
                        int4x2_t *kernel,
                        half *partial,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_partial(float *dst,
                        int4x2_t *src,
                        int4x2_t *kernel,
                        float *partial,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int8_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int16_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(half *dst,
                        int8_t *src,
                        int4x2_t *kernel,
                        half *partial,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_partial(float *dst,
                        int8_t *src,
                        int4x2_t *kernel,
                        float *partial,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int16_t *src,
                         int4x2_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int4x2_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int8_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int16_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int8_t *src,
                         int4x2_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int8_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int16_t *src,
                         int4x2_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int4x2_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

Cambricon@155chb

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int8_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int8_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int4x2_t *src,
                         int16_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         int4x2_t *src,
                         int16_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(half *dst,
                         int8_t *src,
                         int4x2_t *kernel,
                         half *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         int8_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(half *dst,
                        int16_t *src,
                        int4x2_t *kernel,
                        half *partial,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position,
                        int indilation_width,
                        int indilation_height,
                        int outdilation_width,
                        int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         int16_t *src,
                         int4x2_t *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int fix_position,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

```
void __bang_conv_partial(float *dst,
                         bfloat16_t *src,
                         float *kernel,
                         float *partial,
                         int channel_input,
                         int height,
                         int width,
                         int kernel_height,
                         int kernel_width,
                         int stride_width,
                         int stride_height,
                         int channel_output,
                         int indilation_width,
                         int indilation_height,
                         int outdilation_width,
                         int outdilation_height)
```

In `__nram__` address space, uses the four-dimensional convolution kernel `<kernel>[<channel_output>, <kernel_height>, <kernel_width>, <channel_input>]` to perform a partial convolution operation on the three-dimensional tensor `<src>[<height>, <width>, <channel_input>]` with an interactive step size of `[<stride_width>, <stride_height>]`, then adds the three-dimensional tensor `<partial>[<dst_height>, <dst_width>, <channel_output>]`, and stores the result in the three-dimensional tensor `<dst>[<dst_height>, <dst_width>, <channel_output>]`.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination tensor which has $NH_oW_oC_o$ data layout.
- [in] `src`: The address of source tensor which has $NH_iW_iC_i$ data layout.
- [in] `kernel`: The address of filter tensor which has $C_oH_kW_kC_i$ data layout.
- [in] `partial`: The address of partial_sum tensor which has $NH_oW_oC_o$ data layout.
- [in] `channel_input`: Number of input channels.
- [in] `height`: The height of source tensor.
- [in] `width`: The width of source tensor.
- [in] `kernel_height`: The height of filter tensor.
- [in] `kernel_width`: The width of filter tensor.
- [in] `stride_width`: The stride in W direction.
- [in] `stride_height`: The stride in H direction.
- [in] `channel_output`: Number of output channels.
- [in] `fix_position`: Scale Sum of the scale of `<src>` and `<kernel>`.
- [in] `indilation_width`: Input dilation in W direction.
- [in] `indilation_height`: Input dilation in H direction.
- [in] `outdilation_width`: Output dilation in W direction.
- [in] `outdilation_height`: Output dilation in H direction.

Return

- `void`

Remark

- `<src>`, `<dst>` and `<partial>` must point to `__nram__` address space;
- `<kernel>` must point to `__wram__` address space;
- `<fix_position>` is the sum of the scale factor of `<src>` and `<kernel>`;
- `<fix_position>` must be in the range [-127, 127];
- The address of `<dst>`, `<src>` or `<partial>` must be 64-byte aligned on `(m)tp_2xx`;
- The address of `<kernel>` must be 32-byte aligned;
- `<channel_input> * sizeof(typeof<src>)` must be 64-byte aligned on `(m)tp_2xx`;
- `<channel_output>` must be divisible by 64 on `(m)tp_2xx`;
- `<channel_output> * sizeof(typeof<dst>)` must be 64-byte aligned on `(m)tp_2xx`;
- The byte size of `<kernel>` must be 64-byte aligned; otherwise, the non-aligned part of `<kernel>` will be random numbers on `mtp_372` and `tp_322`, and zero on other architecture;
- On `(m)tp_3xx` and higher, `ci` dimension space of `<kernel>` must satisfy the following alignment constraints:
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>) == 4$, $ci * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 16-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>) == 2$, $ci * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 32-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>)$ doesn't satisfy the above mentioned constraints, $ci * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 64-byte aligned;
- `co` dimension space of `<kernel>` must satisfy that `co` is divisible by 64;
- The difference between `__bang_conv_partial` and `__bang_conv` with bias version is: the parameter `<partial>` of `__bang_conv_partial` is a tensor with the same shape as the `<dst>`; and the parameter `<bias>` of `__bang_conv` is a vector whose length is the same with the numbers of channels of `<dst>`;
- `<kernel_height>`, `<kernel_width>`, `<channel_input>` and `<channel_output>` must be greater than 0;

- `<stride_width>` and `<stride_height>` must be in range [1, 1023];
- `<indilation_width>`, `<indilation_height>`, `<outdilation_width>` and `<outdilation_height>` must be in range [1, 1023], and 1 means no dilation;
- If `<indilation_height>` is used, $(\text{height} - ((\text{kernel_height} - 1) * \text{indilation_height} + 1)) / \text{stride_height} + 1$ must be greater than 0; otherwise, $(\text{height} - \text{kernel_height}) / \text{stride_height} + 1$ must be greater than 0;
- If `<indilation_width>` is used, $(\text{width} - ((\text{kernel_width} - 1) * \text{indilation_width} + 1)) / \text{stride_width} + 1$ must be greater than 0; otherwise, $(\text{width} - \text{kernel_width}) / \text{stride_width} + 1$ must be greater than 0;
- On target `(m)tp_322`, `<kernel>` must be `int4`, `int8_t` or `int16_t`;
- On target `(m)tp_322`, if use `<indilation_height>`, $(\text{height} - ((\text{kernel_height} - 1) * \text{indilation_height} + 1)) / \text{stride_height} + 1$ must be in the range [1, 1024] ;
- On target `(m)tp_322`, if use `<dilation_height>`, $(\text{height} - ((\text{kernel_height} - 1) * \text{dilation_height} + 1)) / \text{stride_height} + 1$ must be in the range [1, 1024] ;
- On target `(m)tp_322`, if neither `<dilation_height>` nor `<indilation_height>` are used, $(\text{height} - \text{kernel_height}) / \text{stride_height} + 1$ must be in the range [1, 1024] ;
- `<dst>` cannot be overlapped with `<src>`;
- `<dst>` can be overlapped with `<partial>`.

Compatibility between Various Architectures

Cambricon@155chb

Table 3.18: Conv Partial Data Types Supported on `(m)tp_2xx`

Src Type	Kernel Type	Dst Type	Partial Type
int16	int8	float	float
int8	int8	float	float
int16	int16	float	float
int4	int4	float	float
int8	int4	float	float
int16	int4	float	float
int16	int16	half	half
int16	int8	half	half
int8	int8	half	half
int4	int4	half	half
int8	int4	half	half
int16	int4	half	half

Table 3.19: Conv Partial Data Types Supported on (m)tp_3xx

Src Type	Kernel Type	Dst Type	Partial Type
int16	int8	float	float
int8	int8	float	float
int16	int16	float	float
int16	int16	half	half
int16	int8	half	half
int8	int8	half	half
float	float	float	float
float	half	float	float
half	float	float	float
half	half	float	float
float	bfloat16_t	float	float
bfloat16_t	float	float	float
bfloat16_t	bfloat16_t	float	float
int4	int4	float	float
int4	int8	float	float
int4	int16	float	float
int8	int4	float	float
int16	int4	float	float
int4	int4	half	half
int4	int8	half	half
int4	int16	half	half
int8	int4	half	half
int16	int4	half	half

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_2xx.

Example

```

#include <bang.h>

#define IN_CHANNEL 128
#define IN_HEIGHT 9
#define IN_WIDTH 8
#define FILTER_HEIGHT 2
#define FILTER_WIDTH 3
#define STRIDE_HEIGHT 4
#define STRIDE_WIDTH 3
#define OUT_CHANNEL 64
#define POS 2
#define DILATION_HEIGHT 4
#define DILATION_WIDTH 2

#define NEW_FILTER_HEIGHT ((FILTER_HEIGHT - 1) * DILATION_HEIGHT + 1)
#define NEW_FILTER_WIDTH ((FILTER_WIDTH - 1) * DILATION_WIDTH + 1)
#define OUT_HEIGHT (((IN_HEIGHT) - (NEW_FILTER_HEIGHT)) / (STRIDE_HEIGHT)) + 1
#define OUT_WIDTH (((IN_WIDTH) - (NEW_FILTER_WIDTH)) / (STRIDE_WIDTH)) + 1
#define OUT_DATA_NUM ((OUT_HEIGHT) * (OUT_WIDTH) * (OUT_CHANNEL))
#define IN_DATA_NUM ((IN_HEIGHT) * (IN_WIDTH) * (IN_CHANNEL))
#define FILTER_DATA_NUM ((FILTER_HEIGHT) * (FILTER_WIDTH) *\

(IN_CHANNEL) * (OUT_CHANNEL))

__mlu_entry__ void ConvKernel(float *out_data, int16_t *in_data,
                           int16_t *filter_data, float *partial_data) {
    __nram__ float nram_out_data[OUT_DATA_NUM];
    __nram__ int16_t nram_in_data[IN_DATA_NUM];
    __nram__ float nram_partial_data[OUT_DATA_NUM];
    __wram__ int16_t wram_filter[FILTER_DATA_NUM];

    __memcpy(nram_in_data, in_data, IN_DATA_NUM * sizeof(int16_t), GDRAM2NRAM);
    __memcpy(wram_filter, filter_data, FILTER_DATA_NUM * sizeof(int16_t), GDRAM2WRAM);
    __memcpy(nram_partial_data, partial_data, OUT_DATA_NUM * sizeof(float),
             GDRAM2NRAM);

    __bang_conv_partial(nram_out_data, nram_in_data, wram_filter,
                        nram_partial_data, IN_CHANNEL, IN_HEIGHT, IN_WIDTH,
                        FILTER_HEIGHT, FILTER_WIDTH, STRIDE_WIDTH, STRIDE_HEIGHT,
                        OUT_CHANNEL, POS, DILATION_WIDTH, DILATION_HEIGHT);
    __memcpy(out_data, nram_out_data, OUT_DATA_NUM * sizeof(float), NRAM2GDRAM);
}

```

3.6.7 __bang_conv_partial_tf32

```
void __bang_conv_partial_tf32(float *dst,
                               float *src,
                               float *kernel,
                               float *partial,
                               int channel_input,
                               int height,
                               int width,
                               int kernel_height,
                               int kernel_width,
                               int stride_width,
                               int stride_height,
                               int channel_output)
```

```
void __bang_conv_partial_tf32(float *dst,
                               float *src,
                               float *kernel,
                               float *partial,
                               int channel_input,
                               int height,
                               int width,
                               int kernel_height,
                               int kernel_width,
                               int stride_width,
                               int stride_height,
                               int channel_output,
                               int indilation_width,
                               int indilation_height,
                               int outdilation_width,
                               int outdilation_height)
```

In `__nram__` address space, uses the four-dimensional convolution kernel `<kernel>[<channel_output>, <kernel_height>, <kernel_width>, <channel_input>]` to perform a partial convolution operation on the three-dimensional tensor `<src>[<height>, <width>, <channel_input>]` with an interactive step size of `[<stride_width>, <stride_height>]`, then adds the three-dimensional tensor `<partial>[<dst_height>, <dst_width>, <channel_output>]`, and stores the result in the three-dimensional tensor `<dst>[<dst_height>, <dst_width>, <channel_output>]`.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination tensor which has $NH_oW_oC_o$ data layout.
- [in] `src`: The address of source tensor which has $NH_iW_iC_i$ data layout.
- [in] `kernel`: The address of filter tensor which has $C_oH_kW_kC_i$ data layout.
- [in] `partial`: The address of partial_sum tensor which has $NH_oW_oC_o$ data layout.

- [in] channel_input: Number of input channels.
- [in] height: The height of source tensor.
- [in] width: The width of source tensor.
- [in] kernel_height: The height of filter tensor.
- [in] kernel_width: The width of filter tensor.
- [in] stride_width: The stride in W direction.
- [in] stride_height: The stride in H direction.
- [in] channel_output: Number of output channels.
- [in] indilation_width: Input dilation in W direction.
- [in] indilation_height: Input dilation in H direction.
- [in] outdilation_width: Output dilation in W direction.
- [in] outdilation_height: Output dilation in H direction.

Return

- void

Remark

- <src>, <dst> and <partial> must point to __nram__ address space;
- <kernel> must point to __wram__ address space;
- The address of <kernel> must be 32-byte aligned;
- The byte size of <kernel> must be 64-byte aligned; otherwise, the non-aligned part will be set to zero;
- ci * sizeof(typeof<kernel>) must be 64-byte aligned;
- co dimension space of <kernel> must be divisible by 64;
- <kernel_height>, <kernel_width>, <channel_input> and <channel_output> must be greater than 0;
- <stride_width> and <stride_height> must be in range [1, 1023];
- <indilation_width>, <indilation_height>, <outdilation_width> and <outdilation_height> must be in range [1, 1023], and 1 means no dilation;
- If <indilation_height> is used, (<height> - ((<kernel_height> - 1) * <indilation_height> + 1)) / <stride_height> + 1 must be greater than 0; otherwise, (<height> - <kernel_height>) / <stride_height> + 1 must be greater than 0;
- If <indilation_width> is used, (<width> - ((<kernel_width> - 1) * <indilation_width> + 1)) / <stride_width> + 1 must be greater than 0; otherwise, (<width> - <kernel_width>) / <stride_width> + 1 must be greater than 0;
- <dst> cannot be overlapped with <src>;
- <dst> can be overlapped with <partial>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>
```

```

#define IN_CHANNEL 128
#define IN_HEIGHT 9
#define IN_WIDTH 8
#define FILTER_HEIGHT 2
#define FILTER_WIDTH 3
#define STRIDE_HEIGHT 4
#define STRIDE_WIDTH 3
#define OUT_CHANNEL 64
#define DILATION_HEIGHT 4
#define DILATION_WIDTH 2

#define NEW_FILTER_HEIGHT ((FILTER_HEIGHT - 1) * DILATION_HEIGHT + 1)
#define NEW_FILTER_WIDTH ((FILTER_WIDTH - 1) * DILATION_WIDTH + 1)
#define OUT_HEIGHT (((IN_HEIGHT) - (NEW_FILTER_HEIGHT)) / (STRIDE_HEIGHT)) + 1
#define OUT_WIDTH (((IN_WIDTH) - (NEW_FILTER_WIDTH)) / (STRIDE_WIDTH)) + 1
#define OUT_DATA_NUM ((OUT_HEIGHT) * (OUT_WIDTH) * (OUT_CHANNEL))
#define IN_DATA_NUM ((IN_HEIGHT) * (IN_WIDTH) * (IN_CHANNEL))
#define FILTER_DATA_NUM ((FILTER_HEIGHT) * (FILTER_WIDTH) * (IN_CHANNEL) * (OUT_
→ CHANNEL))

__mlu_entry__ void ConvKernel(float *out_data, float *in_data,
                           float *filter_data, float *partial_data) {
    __nram__ float nram_out_data[OUT_DATA_NUM];
    __nram__ float nram_in_data[IN_DATA_NUM];
    __nram__ float nram_partial_data[OUT_DATA_NUM];
    __wram__ float wram_filter[FILTER_DATA_NUM];

    __memcpy(nram_in_data, in_data, IN_DATA_NUM * sizeof(float), GDRAM2NRAM);
    __memcpy(wram_filter, filter_data, FILTER_DATA_NUM * sizeof(float), GDRAM2WRAM);
    __memcpy(nram_partial_data, partial_data, OUT_DATA_NUM * sizeof(float),
             GDRAM2NRAM);

    __bang_conv_partial_tf32(nram_out_data, nram_in_data, wram_filter,
                            nram_partial_data, IN_CHANNEL, IN_HEIGHT, IN_WIDTH,
                            FILTER_HEIGHT, FILTER_WIDTH, STRIDE_WIDTH, STRIDE_HEIGHT,
                            OUT_CHANNEL, POS, DILATION_WIDTH, DILATION_HEIGHT);
    __memcpy(out_data, nram_out_data, OUT_DATA_NUM * sizeof(float), NRAM2GDRAM);
}

```

3.6.8 __bang_conv_sparse

```
void __bang_conv_sparse(float *dst,
                        float *src,
                        float *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output)
```

```
void __bang_conv_sparse(float *dst,
                        half *src,
                        half *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output)
```

Cambricon@155chb

```
void __bang_conv_sparse(float *dst,
                        int8_t *src,
                        int8_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(float *dst,
                        int8_t *src,
                        int16_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(float *dst,
                        int16_t *src,
                        int8_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(float *dst,
                        int16_t *src,
                        int16_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(half *dst,
                        int8_t *src,
                        int8_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(half *dst,
                        int8_t *src,
                        int16_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(half *dst,
                        int16_t *src,
                        int8_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(half *dst,
                        int16_t *src,
                        int16_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(int16_t *dst,
                        int8_t *src,
                        int8_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(int16_t *dst,
                        int16_t *src,
                        int8_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

```
void __bang_conv_sparse(int16_t *dst,
                        int16_t *src,
                        int16_t *kernel,
                        int channel_input,
                        int height,
                        int width,
                        int kernel_height,
                        int kernel_width,
                        int stride_width,
                        int stride_height,
                        int channel_output,
                        int fix_position)
```

In `__nram__` address space, uses the four-dimensional convolution kernel `<kernel>[<channel_output>, <kernel_height>, <kernel_width>, <channel_input>]` to perform a convolution operation on the three-dimensional tensor `<src>[<height>, <width>, <channel_input>]` in sparse mode with an interactive step size of `[<stride_width>, <stride_height>]`, and stores the result in the three-dimensional tensor `<dst>[<dst_height>, <dst_width>, <channel_output>]`.

Parameters

- [out] `dst`: The address of destination tensor which has $NH_oW_oC_o$ data layout.
- [in] `src`: The address of source tensor which has $NH_iW_iC_i$ data layout.
- [in] `kernel`: The address of filter tensor which has $C_oH_kW_kC_i$ data layout.
- [in] `channel_input`: Number of input channels.
- [in] `height`: The height of source tensor.
- [in] `width`: The width of source tensor.
- [in] `kernel_height`: The height of filter tensor.
- [in] `kernel_width`: The width of filter tensor.
- [in] `stride_width`: The stride in W direction.
- [in] `stride_height`: The stride in H direction.
- [in] `channel_output`: Number of output channels.
- [in] `fix_position`: Sum of scale factor of `<src>` and `<kernel>`.

Return

- `void`

Remark

- In sparse mode, `<kernel>` is in form of 96-byte structure with 64-byte data and 32-byte index. `<kernel>` can either be generated online by `__bang_ssparse_filter_union` related function or be prepared offline by programmers. If programmers want to offline prepare `<kernel>`, please refer to description of the output of `__bang_ssparse_filter_union` function for details on data structure and alignment requirements;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<kernel>` must point to `__wram__` address space;
- `<fix_position>` is the sum of the scale factor of `<src>` and `<kernel>`, and must be in the range [-127, 127];
- The address of `<kernel>` must be 32-byte aligned;

- The byte size of <kernel> must be 64-byte aligned; otherwise, the non-aligned part of <kernel> will be zero;
- ci dimension space of <kernel> must satisfy the following alignment constraints:
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>) == 4$, $\text{ci} * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 16-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>) == 2$, $\text{ci} * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 32-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{kernel}>)$ doesn't satisfy the above mentioned constraints, $\text{ci} * \text{sizeof}(\text{typeof}<\text{kernel}>)$ must be 64-byte aligned;
- co dimension space of <kernel> must be divisible by 64;
- The space size of <bias> is identical to <channel_output>;
- <kernel_height>, <kernel_width>, <channel_input> and <channel_output> must be greater than 0;
- <stride_width> and <stride_height> must be in range [1, 1023];
- $(\text{height} - \text{kernel_height}) / \text{stride_height} + 1$ must be greater than 0;
- $(\text{width} - \text{kernel_width}) / \text{stride_width} + 1$ must be greater than 0;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```
#include "bang.h"
#define IN_CHANNEL 128
#define IN_HEIGHT 2
#define IN_WIDTH 2
#define FILTER_HEIGHT 1
#define FILTER_WIDTH 1
#define STRIDE_HEIGHT 1
#define STRIDE_WIDTH 1
#define OUT_CHANNEL 64
#define LT_NUM 16
#define ESP 1e-7

#define OUT_HEIGHT (((IN_HEIGHT) - (FILTER_HEIGHT)) / (STRIDE_HEIGHT)) + 1
#define OUT_WIDTH (((IN_WIDTH) - (FILTER_WIDTH)) / (STRIDE_WIDTH)) + 1

#define OUT_DATA_NUM ((OUT_HEIGHT) * (OUT_WIDTH) * (OUT_CHANNEL))
#define IN_DATA_NUM ((IN_HEIGHT) * (IN_WIDTH) * (IN_CHANNEL))
#define FILTER_DATA_NUM ((FILTER_HEIGHT) * (FILTER_WIDTH) * (IN_CHANNEL) * (OUT_CHANNEL))
#define FILTER_WITH_INDEX_DATA_NUM (FILTER_DATA_NUM / 128 * 96)

__mlu_global__ void kernel(float* dst, int16_t* src, int16_t* filter) {
    __nram__ float nram_out_data[OUT_DATA_NUM];
}
```

```
__nram__ int16_t nram_in_data[IN_DATA_NUM];
__nram__ int16_t nram_filter[FILTER_DATA_NUM];
__nram__ int16_t nram_filter_with_index[FILTER_WITH_INDEX_DATA_NUM];
__wram__ int16_t wram_filter_with_index[FILTER_WITH_INDEX_DATA_NUM];

__memcpy(nram_in_data, src, IN_DATA_NUM * sizeof(int16_t), GDRAM2NRAM);
__memcpy(nram_filter, filter, FILTER_DATA_NUM * sizeof(int16_t), GDRAM2NRAM);
__bang_ssparse_filter_union(nram_filter_with_index, nram_filter,
                            IN_CHANNEL * sizeof(int16_t), FILTER_HEIGHT,
                            FILTER_WIDTH, OUT_CHANNEL);
__memcpy(wram_filter_with_index, nram_filter_with_index,
        FILTER_WITH_INDEX_DATA_NUM * sizeof(int16_t), NRAM2WRAM);
__bang_conv_sparse(nram_out_data, nram_in_data, wram_filter_with_index,
                    IN_CHANNEL, IN_HEIGHT, IN_WIDTH, FILTER_HEIGHT,
                    FILTER_WIDTH, STRIDE_WIDTH, STRIDE_HEIGHT, OUT_CHANNEL, 0);
__memcpy(dst, nram_out_data, OUT_DATA_NUM * sizeof(float), NRAM2GDRAM);
}
```

3.6.9 __bang_conv_tf32

```
void __bang_conv_tf32(float *dst,
                      float *src,
                      float *kernel,
                      int channel_input,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      int channel_output)
```

```
void __bang_conv_tf32(float *dst,
                      float *src,
                      float *kernel,
                      int channel_input,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      int channel_output,
                      int indilation_width,
                      int indilation_height,
                      int outdilation_width,
                      int outdilation_height)
```

In `__nram__` address space, uses the four-dimensional convolution kernel `<kernel>[<channel_output>, <kernel_height>, <kernel_width>, <channel_input>]` to perform a convolution operation on the three-dimensional tensor `<src>[<height>, <width>, <channel_input>]` with an interactive step size of `[<stride_width>, <stride_height>]`, and stores the result in the three-dimensional tensor `<dst>[<dst_height>, <dst_width>, <channel_output>]`.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination tensor which has $NH_oW_oC_o$ data layout.
- [in] `src`: The address of source tensor which has $NH_iW_iC_i$ data layout.
- [in] `kernel`: The address of filter tensor which has $C_oH_kW_kC_i$ data layout.
- [in] `channel_input`: Number of input channels.
- [in] `height`: The height of source tensor.
- [in] `width`: The width of source tensor.
- [in] `kernel_height`: The height of filter tensor.
- [in] `kernel_width`: The width of filter tensor.
- [in] `stride_width`: The stride in W direction.
- [in] `stride_height`: The stride in H direction.
- [in] `channel_output`: Number of output channels.
- [in] `indilation_width`: Input dilation in W direction.
- [in] `indilation_height`: Input dilation in H direction.
- [in] `outdilation_width`: Output dilation in W direction.
- [in] `outdilation_height`: Output dilation in H direction.

Return

- `void`.

Remark

- `<src>`, `<dst>` must point to `__nram__` address space;
- `<kernel>` must point to `__wram__` address space;

- The address of <kernel> must be 32-byte aligned;
- The byte size of <kernel> must be 64-byte aligned; otherwise, the non-aligned part will be set to zero;
- ci * sizeof(typeof<kernel>) must be 64-byte aligned;
- co dimension space of <kernel> must be divisible by 64;
- <kernel_height>, <kernel_width>, <channel_input> and <channel_output> must be greater than 0;
- <stride_width> and <stride_height> must be in range [1, 1023];
- <indilation_width>, <indilation_height>, <outdilation_width> and <outdilation_height> must be in range [1, 1023], and 1 means no dilation;
- If <indilation_height> is used, (<height> - ((<kernel_height> - 1) * <indilation_height> + 1)) / <stride_height> + 1 must be greater than 0; otherwise, (<height> - <kernel_height>) / <stride_height> + 1 must be greater than 0;
- If <indilation_width> is used, (<width> - ((<kernel_width> - 1) * <indilation_width> + 1)) / <stride_width> + 1 must be greater than 0; otherwise, (<width> - <kernel_width>) / <stride_width> + 1 must be greater than 0;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bang_conv_partial_tf32](#) for more detail.

3.6.10 __bang_maxpool

```
void __bang_maxpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width)
```

```
void __bang_maxpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_maxpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_maxpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_maxpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_maxpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_maxpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

Applies maxpooling forward operation on <src> [`<height>, <width>, <channel>`], a three-dimensional tensor with sliding window [`<kernel_height>, <kernel_width>`] and stride [`<stride_width>, <stride_height>`], and selects the maximum value in each window. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination tensor. And the tensor data layout is HWC.
- [in] src: The address of source tensor. And the tensor data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of input feature map.
- [in] width: The width of input feature map.
- [in] kernel_height: The height of sliding window.
- [in] kernel_width: The width of sliding window.
- [in] stride_width: Stride of sliding window in W direction.
- [in] stride_height: Stride of sliding window in H direction.
- [in] in_dh: Dilation in H direction of input.
- [in] in_dw: Dilation in W direction of input.
- [in] out_dh: Dilation in H direction of output.
- [in] out_dw: Dilation in W direction of output.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The function with dilation is supported on (m)tp_3xx or higher;
- <dst> cannot be overlapped with <src>;
- In the function without stride, <stride_width> equals <kernel_width> and <stride_height> equals <kernel_height>;
- (*kernel_height* × *kernel_width*) < 2¹⁶;
- <channel>, <stride_height>, <stride_width>, <in_dh>, <in_dw>, <out_dh> and <out_dw> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define CHANNELS 64
#define HEIGHT 4
#define WIDTH 4
#define KERNEL_HEIGHT 2
#define KERNEL_WIDTH 2
#define BOTTOM_DATA_COUNT ((CHANNELS) * (WIDTH) * (HEIGHT))
#define TOP_DATA_COUNT \
    ((CHANNELS) * (HEIGHT / KERNEL_HEIGHT) * (WIDTH / KERNEL_WIDTH))
```

```
--mlu_entry__ void maxPoolingKernel(half* bottom_data, half* top_data,
                                    int channels, int height, int width,
                                    int pooled_height, int pooled_width) {
    __nram__ half a_tmp[BOTTOM_DATA_COUNT];
    __nram__ half b_tmp[TOP_DATA_COUNT];
    __memcpy(a_tmp, bottom_data, BOTTOM_DATA_COUNT * sizeof(half), GDRAM2NRAM);
    __bang_maxpool(b_tmp, a_tmp, CHANNELS, HEIGHT, WIDTH, KERNEL_HEIGHT, KERNEL_WIDTH);
    __memcpy(top_data, b_tmp, TOP_DATA_COUNT * sizeof(half), NRAM2GDRAM);
}
```

3.6.11 __bang_maxpool_bp

```
void __bang_maxpool_bp(half*dst,
                      half*src,
                      short *mask,
                      int channel,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      mluPoolBPOverlap overlap = OVERLAP_ACC)
```

```
void __bang_maxpool_bp(bfloat16_t *dst,
                      bfloat16_t *src,
                      unsigned short *mask,
                      int channel,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      mluPoolBPOverlap overlap = OVERLAP_ACC)
```

```
void __bang_maxpool_bp(float *dst,
                      float *src,
                      int *mask,
                      int channel,
                      int height,
                      int width,
                      int kernel_height,
                      int kernel_width,
                      int stride_width,
                      int stride_height,
                      mluPoolBPOverlap overlap = OVERLAP_ACC)
```

Applies maxpooling backward propagation operation on <src> [, ,], a three-dimensional tensor, with sliding window [,] and stride [,], and selects the index with the maximum value. <overlap> indicates the type of overlap options. <overlap> is assigned to an enumerated type called `mluPoolBPOverlap` that contains 2 enumerators listed in the table below. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded.

Table 3.20: Semantics of `mluPoolBPOverlap`

<code>mluPoolBPOverlap</code> Type	Semantic
<code>OVERLAP_ACC</code>	Accumulates the overlap parts of the output.
<code>OVERLAP_COVER</code>	Covers the overlap parts of the output.

Parameters

- [out] `dst`: The address of destination tensor whose data layout is HWC.
- [in] `src`: The address of source tensor whose data layout is HWC.
- [in] `mask`: The index of the maximum value inside a kernel, and the tensor data layout is HWC.
- [in] `channel`: Input channel.
- [in] `height`: The height of output feature map.
- [in] `width`: The width of output feature map.
- [in] `kernel_height`: The height of kernel.
- [in] `kernel_width`: The width of kernel.
- [in] `stride_width`: Stride of sliding window in W direction.
- [in] `stride_height`: Stride of sliding window in H direction.
- [in] `overlap`: The type of overlap options.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- `<channel> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;

- When the data type of <src> and <dst> is half or bfloat16_t, the data type of <mask> must be short. When the data type of <src> and <dst> are float, the data type of <mask> must be int;
- <dst> cannot be overlapped with <src>;
- The default mluPoolBpOverlap option is OVERLAP_ACC;
- <channel>, <stride_height> and <stride_width> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void PoolMaxBpKernel(half* output, half* input, int16_t* mask,
                                    int channels, int out_height,
                                    int out_width, int kernel_height,
                                    int kernel_width, int stride_width,
                                    int stride_height) {
    __nram__ half a_tmp[INPUT_COUNT];
    __nram__ half b_tmp[OUTPUT_COUNT];
    __nram__ int16_t c_tmp[INPUT_COUNT];

    __memcpy(b_tmp, output, OUTPUT_COUNT * sizeof(half), GDRAM2NRAM);
    __memcpy(a_tmp, input, INPUT_COUNT * sizeof(half), GDRAM2NRAM);
    __memcpy(c_tmp, mask, INPUT_COUNT * sizeof(int16_t), GDRAM2NRAM);

    __bang_maxpool_bp(b_tmp, a_tmp, c_tmp, channels, out_height, out_width,
                      kernel_height, kernel_width, stride_width, stride_height);

    __memcpy(output, b_tmp, OUTPUT_COUNT * sizeof(half), NRAM2GDRAM);
}
```

3.6.12 __bang_maxpool_index

```
void __bang_maxpool_index(unsigned short *dst,
                          half *src,
                          int channel,
                          int height,
                          int width,
                          int kernel_height,
                          int kernel_width)
```

```
void __bang_maxpool_index(unsigned short *dst,
                           half *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height)
```

```
void __bang_maxpool_index(unsigned short *dst,
                           bfloat16_t *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height)
```

```
void __bang_maxpool_index(unsigned short *dst,
                           half *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height,
                           int in_dh,
                           int in_dw,
                           int out_dh,
                           int out_dw)
```

Cambricon@155chb

```
void __bang_maxpool_index(unsigned int *dst,
                           float *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height,
                           int in_dh,
                           int in_dw,
                           int out_dh,
                           int out_dw)
```

```
void __bang_maxpool_index(unsigned short *dst,
                           bfloat16_t *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height,
                           int in_dh,
                           int in_dw,
                           int out_dh,
                           int out_dw)
```

```
void __bang_maxpool_index(unsigned int *dst,
                           float *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height)
```

Applies maxpooling forward operation with index on <src> [<height>, <width>, <channel>], a three-dimensional tensor, with sliding window [<kernel_height>, <kernel_width>] and stride [<stride_width>, <stride_height>], and selects the index of the maximum value in each window. When the data type of <src> is half or bfloat16_t, the data type of <dst> is unsigned short. When the data type of <src> is float, the data type of <dst> is unsigned int. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for

accuracy information.

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of input feature map.
- [in] width: The width of input feature map.
- [in] kernel_height: The height of kernel.
- [in] kernel_width: The width of kernel.
- [in] stride_width: Stride of sliding window in W direction.
- [in] stride_height: Stride of sliding window in H direction.
- [in] in_dh: Dilation in H direction of input.
- [in] in_dw: Dilation in W direction of input.
- [in] out_dh: Dilation in H direction of output.
- [in] out_dw: Dilation in W direction of output.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The function with dilation is supported on (m)tp_3xx or higher;
- <dst> cannot be overlapped with <src>;
- In the function without stride, <stride_width> equals <kernel_width> and <stride_height> equals <kernel_height>;
- ($\text{kernel_height} \times \text{kernel_width}$) $< 2^{16}$;
- <channel>, <stride_height>, <stride_width>, <in_dh>, <in_dw>, <out_dh> and <out_dw> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ ≥ 200 ;
- CNCC Version: cncc --version $\geq 2.8.0$;
- Cambricon BANG Compute Arch Version: cncc --bang-arch $\geq \text{compute_20}$;
- MLU Compute Arch Version: cncc --bang-mlu-arch $\geq (\text{m})\text{tp_2xx}$.

Example

```
#include <bang.h>

#define CHANNELS 64
#define HEIGHT 9
#define WIDTH 12
#define KERNEL_HEIGHT 5
#define KERNEL_WIDTH 4
```

```

#define BOTTOM_DATA_COUNT ((CHANNELS) * (WIDTH) * (HEIGHT))
#define TOP_DATA_COUNT \
((CHANNELS) * (HEIGHT / KERNEL_HEIGHT) * (WIDTH / KERNEL_WIDTH))

__mlu_entry__ void MaxPoolIndexKernel(half* bottom_data, int16_t* top_data,
                                      int channels, int height, int width,
                                      int pooled_height, int pooled_width) {
    __nram__ half a_tmp[BOTTOM_DATA_COUNT];
    __nram__ uint16_t b_tmp[TOP_DATA_COUNT];
    __memcpy(a_tmp, bottom_data, BOTTOM_DATA_COUNT * sizeof(half), GDRAM2NRAM);
    __bang_maxpool_index(b_tmp, a_tmp, CHANNELS, HEIGHT, WIDTH,
                         KERNEL_HEIGHT, KERNEL_WIDTH);
    __memcpy(top_data, b_tmp, TOP_DATA_COUNT * sizeof(int16_t), NRAM2GDRAM);
}

```

3.6.13 __bang_maxpool_value_index

```

void __bang_maxpool_value_index(half *dst,
                                half *src,
                                int channel,
                                int height,
                                int width,
                                int kernel_height,
                                int kernel_width,
                                int stride_width,
                                int stride_height,
                                int value_index_stride,
                                int in_dh,
                                int in_dw,
                                int out_dh,
                                int out_dw)

```

```

void __bang_maxpool_value_index(bfloat16_t *dst,
                                bfloat16_t *src,
                                int channel,
                                int height,
                                int width,
                                int kernel_height,
                                int kernel_width,
                                int stride_width,
                                int stride_height,
                                int value_index_stride,
                                int in_dh,
                                int in_dw,
                                int out_dh,
                                int out_dw)

```

```
void __bang_maxpool_value_index(float *dst,
                                float *src,
                                int channel,
                                int height,
                                int width,
                                int kernel_height,
                                int kernel_width,
                                int stride_width,
                                int stride_height,
                                int value_index_stride,
                                int in_dh,
                                int in_dw,
                                int out_dh,
                                int out_dw)
```

- Performs maximum pooling operation with the kernel [kernel_height, kernel_width] and the stride [stride_width, stride_height] on the tensor <src>[channel, height, width] in __nram__ address space, then stores the maximum value and the corresponding index to <dst>. When the data type of <src> is half, the data type of value in <dst> is half and the data type of index in <dst> is unsigned short. When the data type of <src> is float, the data type of value in <dst> is float and the data type of index in <dst> is unsigned int. When the kernel moves to the edge of input <width> and the kernel overflows, it will automatically adapt to the size of <kernel_width>, so as to ensure that useless input is not included. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: The input channel.
- [in] height: The height of input source.
- [in] width: The width of input source.
- [in] kernel_height: The height of kernel.
- [in] kernel_width: The width of kernel.
- [in] stride_width: The stride in W direction.
- [in] stride_height: The stride in H direction.
- [in] value_index_stride: The offset from the beginning of data value to index value in bytes.
- [in] in_dh: Dilation in H direction of input.
- [in] in_dw: Dilation in W direction of input.
- [in] out_dh: Dilation in H direction of output.
- [in] out_dw: Dilation in W direction of output.

Return

- void.

Remark

- <dst> and <src> must point to __nram__ address space;

- For the output tensor $\langle\text{dst}\rangle[\text{output_channel}, \text{output_height}, \text{output_width}]$, $[\text{output_channel}] = \langle\text{channel}\rangle$, and $[\text{output_height}, \text{output_width}]$ are obtained by deducing from $[\text{height}, \text{width}]$, $[\text{kernel_height}, \text{kernel_width}]$ and $[\text{stride_width}, \text{stride_height}]$;
- $[\text{output_height}]$ and $[\text{output_width}]$ of $\langle\text{dst}\rangle$ must be greater than 0;
- $(\text{kernel_height} \times \text{kernel_width}) < 2^{16}$;
- For the value part of $\langle\text{dst}\rangle$, when $\langle\text{value_index_stride}\rangle > 0$, the address operands $\langle\text{dst}\rangle$ can be overlapped with $\langle\text{src}\rangle$;
- $\langle\text{channel}\rangle$, $\langle\text{stride_height}\rangle$, $\langle\text{stride_width}\rangle$, $\langle\text{in_dh}\rangle$, $\langle\text{in_dw}\rangle$, $\langle\text{out_dh}\rangle$ and $\langle\text{out_dw}\rangle$ must be greater than 0;
- $\langle\text{kernel_height}\rangle$ and $\langle\text{kernel_width}\rangle$ must be greater than 0;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

```
#include <bang.h>

#define DATA_NUM 64

Cambricon@155chb

__mlu_entry__ void kernel(half *dst, half *src, float *dst_f,
                         float *src_f, int channel,
                         int height, int width,
                         int kernel_height, int kernel_width,
                         int stride_width, int stride_height,
                         int value_index_stride,
                         int in_dh, int in_dw,
                         int out_dh, int out_dw) {
    __nram__ half svc_dst[DATA_NUM];
    __nram__ half svc_src[DATA_NUM];
    int datasize = DATA_NUM * sizeof(half);
    __memcpy(svc_src, src, datasize, GDRAM2NRAM);
    __bang_maxpool_value_index(svc_dst, svc_src, channel, height,
                               width, kernel_height, kernel_width,
                               stride_width, stride_height,
                               value_index_stride,
                               in_dh, in_dw, out_dh, out_dw);
    __memcpy(dst, svc_dst, datasize, NRAM2GDRAM);
}
```

3.6.14 __bang_minpool

```
void __bang_minpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width)
```

```
void __bang_minpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_minpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_minpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_minpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_minpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height,
                     int in_dh,
                     int in_dw,
                     int out_dh,
                     int out_dw)
```

```
void __bang_minpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

Applies minpooling forward operation on `<src>``[``<height>, <width>, <channel>]`, a three-dimensional tensor, with sliding window [`<kernel_height>, <kernel_width>`] and stride [`<stride_width>, <stride_height>`], and selects the minimum value in each window. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of input feature map.
- [in] width: The width of input feature map.
- [in] kernel_height: The height of sliding window.
- [in] kernel_width: The width of sliding window.
- [in] stride_width: Stride of sliding window in W direction.
- [in] stride_height: Stride of sliding window in H direction.
- [in] in_dh: Dilation in H direction of input.
- [in] in_dw: Dilation in W direction of input.
- [in] out_dh: Dilation in H direction of output.
- [in] out_dw: Dilation in W direction of output.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The function with dilation is supported on (m)tp_3xx or higher;
- <dst> cannot be overlapped with <src>;
- In the function without stride, <stride_width> equals <kernel_width> and <stride_height> equals <kernel_height>;
- (*kernel_height* × *kernel_width*) < 2¹⁶;
- <channel>, <stride_height>, <stride_width>, <in_dh>, <in_dw>, <out_dh> and <out_dw> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.6.15 __bang_minpool_index

```
void __bang_minpool_index(unsigned short *dst,
                           half *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width)
```

```
void __bang_minpool_index(unsigned short *dst,
                           half *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height)
```

```
void __bang_minpool_index(unsigned short *dst,
                           bfloat16_t *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height)
```

```
void __bang_minpool_index(unsigned short *dst,
                           half *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height,
                           int in_dh,
                           int in_dw,
                           int out_dh,
                           int out_dw)
```

Cambricon@155chb

```
void __bang_minpool_index(unsigned int *dst,
                           float *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height,
                           int in_dh,
                           int in_dw,
                           int out_dh,
                           int out_dw)
```

```
void __bang_minpool_index(unsigned short *dst,
                           bfloat16_t *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height,
                           int in_dh,
                           int in_dw,
                           int out_dh,
                           int out_dw)
```

```
void __bang_minpool_index(unsigned int *dst,
                           float *src,
                           int channel,
                           int height,
                           int width,
                           int kernel_height,
                           int kernel_width,
                           int stride_width,
                           int stride_height)
```

Applies minpooling forward operation with index on `<src>` [`<height>`, `<width>`, `<channel>`], a three-dimensional tensor, with sliding window [`<kernel_height>`, `<kernel_width>`] and stride [`<stride_width>`, `<stride_height>`], and selects the index with the minimum value in each window. When the data type of `<src>` is `half` or `bfloat16_t`, the data type of `<dst>` is `unsigned short`. When the data type of `<src>` is `float`, the data type of `<dst>` is `unsigned int`. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of input feature map.
- [in] width: The width of input feature map.
- [in] kernel_height: The height of sliding window.
- [in] kernel_width: The width of sliding window.
- [in] stride_width: Stride of sliding window in W direction.
- [in] stride_height: Stride of sliding window in H direction.
- [in] in_dh: Dilation in H direction of input.
- [in] in_dw: Dilation in W direction of input.
- [in] out_dh: Dilation in H direction of output.
- [in] out_dw: Dilation in W direction of output.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The function with dilation is supported on (m)tp_3xx or higher;
- <dst> cannot be overlapped with <src>;
- In the function without stride, <stride_width> equals <kernel_width> and <stride_height> equals <kernel_height>;
- (*kernel_height* × *kernel_width*) < 2¹⁶;
- <channel>, <stride_height>, <stride_width>, <in_dh>, <in_dw>, <out_dh> and <out_dw> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.6.16 __bang_minpool_value_index

```
void __bang_minpool_value_index(half *dst,
                                half *src,
                                int channel,
                                int height,
                                int width,
                                int kernel_height,
                                int kernel_width,
                                int stride_width,
                                int stride_height,
                                int value_index_stride,
                                int in_dh,
                                int in_dw,
                                int out_dh,
                                int out_dw)
```

```
void __bang_minpool_value_index(bfloat16_t *dst,
                                bfloat16_t *src,
                                int channel,
                                int height,
                                int width,
                                int kernel_height,
                                int kernel_width,
                                int stride_width,
                                int stride_height,
                                int value_index_stride,
                                int in_dh,
                                int in_dw,
                                int out_dh,
                                int out_dw)
```

```
void __bang_minpool_value_index(float *dst,
                                float *src,
                                int channel,
                                int height,
                                int width,
                                int kernel_height,
                                int kernel_width,
                                int stride_width,
                                int stride_height,
                                int value_index_stride,
                                int in_dh,
                                int in_dw,
                                int out_dh,
                                int out_dw)
```

- Performs minimum pooling operation with the kernel [`<kernel_height>`, `<kernel_width>`] and the stride [`<stride_width>`, `<stride_height>`] on the tensor `<src>``[``<channel>, <height>, <width>]` in `__nram__` address space, and stores the minimum value and the corresponding index to `<dst>`. When the data type of `<src>` is `half` or `bfloat16_t`, the data type of `value` in `<dst>` is `half` and the data type of `index` in `<dst>` is `unsigned short`. When the data type of `<src>` is `float`, the data type of `value` in `<dst>` is `float` and the data type of `index` in `<dst>` is `unsigned int`. When the kernel moves to the edge of input `<width>` and the kernel overflows, it will automatically adapt to the size of `<kernel_width>`, so as to ensure that useless input is not included. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination tensor whose data layout is HWC.
- [in] `src`: The address of source tensor whose data layout is HWC.
- [in] `channel`: The input channel.
- [in] `height`: The height of input source.
- [in] `width`: The width of input source.
- [in] `kernel_height`: The height of kernel.
- [in] `kernel_width`: The width of kernel.
- [in] `stride_width`: The stride in W direction.
- [in] `stride_height`: The stride in H direction.
- [in] `value_index_stride`: The offset from the beginning of data value to index value in bytes.
- [in] `in_dh`: Dilation in H direction of input.
- [in] `in_dw`: Dilation in W direction of input.
- [in] `out_dh`: Dilation in H direction of output.
- [in] `out_dw`: Dilation in W direction of output.

Return

- `void`.

Remark

- `<dst>` and `<src>` must point to `__nram__` address space;
- `bfloat16_t` is supported on (m)tp_5xx or higher;
- For the output tensor `<dst>[output_channel, output_height, output_width]`, `[output_channel] = <channel>`, and `[output_height, output_width]` are obtained by deducing from `[height, width]`, `[kernel_height, kernel_width]` and `[stride_width, stride_height]`;
- `[output_height]` and `[output_width]` of `<dst>` must be greater than 0;
- $(kernel_height \times kernel_width) < 2^{16}$;
- For the value part of `<dst>`, when `<value_index_stride> > 0`, the address operands `<dst>` can be overlapped with `<src>`;
- `<channel>`, `<stride_height>`, `<stride_width>`, `<in_dh>`, `<in_dw>`, `<out_dh>` and `<out_dw>` must be greater than 0;
- `<kernel_height>` and `<kernel_width>` must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;

- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define DATA_NUM 64

__mlu_entry__ void kernel(half *dst, half *src, float *dst_f, float *src_f,
                         int channel, int height, int width, int kernel_height,
                         int kernel_width, int stride_width, int stride_height,
                         int value_index_stride, int in_dh, int in_dw,
                         int out_dh, int out_dw);

int datasize = DATA_NUM * sizeof(half);
__nram__ half svc_dst[DATA_NUM];
__nram__ half svc_src[DATA_NUM];
__memcpy(svc_src, src, datasize, GDRAM2NRAM);
__bang_minpool_value_index(svc_dst, svc_src, channel, height, width,
                           kernel_height, kernel_width, stride_width,
                           stride_height, value_index_stride,
                           in_dh, in_dw, out_dh, out_dw);
__memcpy(dst, svc_dst, datasize, NRAM2GDRAM);
}
```

Cambricon@155chb

3.6.17 __bang_mlp

```
void __bang_mlp(float *dst,
                int32_t *src,
                float *bias,
                int16_t *filter,
                const int height,
                const int width,
                int fix_position)
```

```
void __bang_mlp(half *dst,
                int8_t *src,
                int8_t *filter,
                int height,
                int width,
                int fix_position)
```

```
void __bang_mlp(half *dst,
                 int16_t *src,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(half *dst,
                 int16_t *src,
                 int16_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(float *dst,
                 int8_t *src,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(float *dst,
                 int16_t *src,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(float *dst,
                 int16_t *src,
                 int16_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(int16_t *dst,
                 int8_t *src,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(int16_t *dst,
                 int16_t *src,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(int16_t *dst,
                 int16_t *src,
                 int16_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(half *dst,
                 int8_t *src,
                 half *bias,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(half *dst,
                 int16_t *src,
                 half *bias,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(half *dst,
                 int16_t *src,
                 half *bias,
                 int16_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(float *dst,
                 int8_t *src,
                 float *bias,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(float *dst,
                 int16_t *src,
                 float *bias,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(float *dst,
                 int16_t *src,
                 float *bias,
                 int16_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(int16_t *dst,
                 int8_t *src,
                 int16_t *bias,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(int16_t *dst,
                 int16_t *src,
                 int16_t *bias,
                 int8_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

```
void __bang_mlp(int16_t *dst,
                 int16_t *src,
                 int16_t *bias,
                 int16_t *filter,
                 int height,
                 int width,
                 int fix_position)
```

Applies multilayer perception operation. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

For version with <bias>: $<dst> = <src> \times <filter> + <bias>$.

For version without <bias>: $<dst> = <src> \times <filter>$.

Parameters

- [out] `dst`: The address of destination matrix.
- [in] `src`: The address of source matrix.
- [in] `bias`: The address of bias matrix.
- [in] `filter`: The address of filter matrix.
- [in] `height`: The height of `<filter>`.
- [in] `width`: The width of `<filter>`.
- [in] `fix_position`: Sum of scale factor of `<src>` and `<kernel>`.

Return

- `void`.

Remark

- `<dst>`, `<src>` and `<bias>` must point to `__nram__` address space;
- `<filter>` must point to `__wram__` address space;
- `<fix_position>` must be in the range `[-127, 127]`;
- The address of `<dst>`, `<src>` and `<bias>` must be 64-byte aligned on `(m)tp_2xx`;
- The address of `<filter>` must be 32-byte aligned;
- `<height>` must be divisible by 64 on `(m)tp_2xx`;
- `<height> * sizeof(typeof<dst>)` must be 64-byte aligned on `(m)tp_2xx`;
- `<width> * sizeof(typeof<src>)` must be 64-byte aligned on `(m)tp_2xx`;
- `__bang_mlp` of int input type version is not recommended, because it is implemented by combination, with lower performance. `__clang_bang_extra.h` should be included when using this version;
- On `(m)tp_3xx`, if `--bang-wram-align64` option is used, the byte size of `<filter>` must be 64-byte aligned; otherwise, the byte size of `<filter>` must be 32-byte aligned;
- On `(m)tp_3xx`, width dimension space of `<filter>` must satisfy the following alignment constraints:
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{filter}>) == 4$, width * $\text{sizeof}(\text{typeof}<\text{filter}>)$ must be 16-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{filter}>) == 2$, width * $\text{sizeof}(\text{typeof}<\text{filter}>)$ must be 32-byte aligned;
 - If $\text{sizeof}(\text{typeof}<\text{src}>) / \text{sizeof}(\text{typeof}<\text{filter}>)$ doesn't satisfy the above mentioned constraints, width * $\text{sizeof}(\text{typeof}<\text{filter}>)$ must be 64-byte aligned;
- height dimension space of `<filter>` must satisfy that `height` is divisible by 64 on `(m)tp_3xx`;
- On `(m)tp_3xx`, if `<kernel>` does not follow the above alignment rules, the values of the non-aligned part will be random numbers, which may cause calculation errors;
- `<width>` and `<height>` must be greater than 0;
- `<dst>` cannot be overlapped with `<src>`;
- `<dst>` cannot be overlapped with `<bias>`;

Compatibility between Various Architectures

Table 3.21: Data Types Supported on (m)tp_2xx

Dst Type	Src Type	Bias Type	Filter Type
half	int16	half	int16
half	int8	half	int8
half	int16	half	int8
float	int16	float	int16
float	int8	float	int8
float	int16	float	int8
float	int	float	int16
int16	int16	int16	int16
int16	int8	int16	int8
int16	int16	int16	int8
half	int8	none	int8
half	int16	none	int8
half	int16	none	int16
float	int8	none	int8
float	int16	none	int8
float	int16	none	int16
int16	int8	none	int8
int16	int16	none	int8
int16	int16	none	int16

Table 3.22: mlp Data Types Supported on (m)tp_3xx

Dst Type	Src Type	Bias Type	Filter Type
half	int16	half	int16
half	int8	half	int8
half	int16	half	int8
float	int16	float	int16
float	int8	float	int8

continues on next page

Table 3.22 – continued from previous page

Dst Type	Src Type	Bias Type	Filter Type
float	int16	float	int8
float	int	float	int16
half	int8	none	int8
half	int16	none	int8
half	int16	none	int16
float	int8	none	int8
float	int16	none	int8
float	int16	none	int16

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define HEIGHT 64
#define WIDTH 64
#define POS 0

__mlu_entry__ void MlpKernel(half* out_data, int16_t* in_data,
                            int16_t* filter_data, half* bias_data,
                            int height, int width, int pos) {
    __nram__ half nram_out_data[HEIGHT];
    __nram__ half nram_bias_data[HEIGHT];
    __nram__ int16_t nram_in_data[WIDTH];
    __wram__ int16_t wram_filter[HEIGHT * WIDTH];
    __memcpy(nram_in_data, in_data, WIDTH * sizeof(int16_t), GDRAM2NRAM);
    __memcpy(nram_bias_data, bias_data, HEIGHT * sizeof(half), GDRAM2NRAM);
    __memcpy(wram_filter, filter_data, HEIGHT * WIDTH * sizeof(int16_t),
             GDRAM2WRAM);
    __bang_mlp(nram_out_data, nram_in_data, nram_bias_data, wram_filter,
               HEIGHT, WIDTH, POS);
    __memcpy(out_data, nram_out_data, HEIGHT * sizeof(half), NRAM2GDRAM);
}
```

3.6.18 __bang_reshape_filter

```
void __bang_reshape_filter(half *dst,
                           half *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(short *dst,
                           short *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(unsigned short *dst,
                           unsigned short *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(int8_t *dst,
                           int8_t *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(char *dst,
                           char *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(unsigned char *dst,
                           unsigned char *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(float *dst,
                           float *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(int *dst,
                           int *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

```
void __bang_reshape_filter(unsigned int *dst,
                           unsigned int *src,
                           int n,
                           int h,
                           int w,
                           int c)
```

Reshapes the input kernel suit for MLP and CONV, <n> changed to <n>/64 along <n> direction.

Parameters

- [out] dst: The address of output kernel.
- [in] src: The address of input kernel.
- [in] n: The batch number of input kernel.
- [in] h: The height of input kernel.
- [in] w: The width of input kernel.
- [in] c: The channel input of kernel.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <n> must be divisible by 64;
- < h > × < w > × < c > × sizeof(type) must be divisible by 128;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.6.19 __bang_reshape_nchw2nhwc

```
void __bang_reshape_nchw2nhwc(half *dst,
                               half *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(short *dst,
                               short *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(unsigned short *dst,
                               unsigned short *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(int8_t *dst,
                               int8_t *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(char *dst,
                               char *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(unsigned char *dst,
                               unsigned char *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(float *dst,
                               float *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(int *dst,
                               int *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nchw2nhwc(unsigned int *dst,
                               unsigned int *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

Reshapes the data layout of the kernel from NCHW to NHWC.

Parameters

- [out] dst: The address of output kernel.
- [in] src: The address of input kernel.
- [in] n: The batch number of input kernel.
- [in] h: The height of input kernel.
- [in] w: The width of input kernel.
- [in] c: The channel input of kernel.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- < h > × < w > × sizeof(type) must be divisible by 64;
- < c > × sizeof(type) must be divisible by 64;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.6.20 __bang_reshape_nhwc2nchw

```
void __bang_reshape_nhwc2nchw(half *dst,
                               half *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(short *dst,
                               short *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(unsigned short *dst,
                               unsigned short *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(int8_t *dst,
                               int8_t *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(char *dst,
                               char *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(unsigned char *dst,
                               unsigned char *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(float *dst,
                               float *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(int *dst,
                               int *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

```
void __bang_reshape_nhwc2nchw(unsigned int *dst,
                               unsigned int *src,
                               int n,
                               int h,
                               int w,
                               int c)
```

Reshapes the data layout of the kernel from NHWC to NCHW.

Parameters

- [out] dst: The address of output kernel.
- [in] src: The address of input kernel.
- [in] n: The batch number of input kernel.
- [in] h: The height of input kernel.
- [in] w: The width of input kernel.
- [in] c: The channel input of kernel.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- < h > × < w > × sizeof(type) must be divisible by 64;
- < c > × sizeof(type) must be divisible by 64;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.6.21 __bang_ssparse_filter_index

```
void __bang_ssparse_filter_index(half *dst,
                                 half *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_index(float *dst,
                                 float *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_index(int8_t *dst,
                                 int8_t *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_index(int16_t *dst,
                                 int16_t *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_index(int *dst,
                                 int *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_index(bfloat16_t *dst,
                                 bfloat16_t *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

Selects 2 elements with larger absolute value from every 4 elements of <src> and stores the index of selected elements in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] input_channel: The number of input channel of source vector.
- [in] src_height: The height of source vector.
- [in] src_width: The width of source vector.
- [in] output_channel: The number of channel of destination vector.

Return

- void

Remark

- <dst> and <src> must point to __nram__ address space;
- <src> is split by 8 elements on input_channel dimension. When the split part is less than 8 elements, hardware will pad to 8 elements with zero;
- Each bit in index corresponds to one element in <src>;
- There is no alignment constraint for parameter <input_channel>, but the size of input_channel dimension of <dst> must align to ceil(<input_channel> / (8 * sizeof(typeof(<src>)))) byte(s).

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```
#include "bang.h"

#define SRC_CHANNEL_BYTES (2 * 128)
#define KERNEL_HEIGHT 2
#define KERNEL_WIDTH 3
#define DST_CHANNEL 4
#define INPUT_BYTES \
    ((SRC_CHANNEL_BYTES) * (KERNEL_HEIGHT) * (KERNEL_WIDTH) * (DST_CHANNEL))
#define OUTPUT_BYTES ((INPUT_BYTES) / 4)
#define TOTAL_INPUT_BYTES ((INPUT_BYTES)*2)
#define TOTAL_OUTPUT_BYTES ((OUTPUT_BYTES)*2)

__mlu_global__ void kernel_ssparse_filter_index(bfloat16_t* dst, bfloat16_t* src) {
    __nram__ bfloat16_t dst_nram[TOTAL_OUTPUT_BYTES / sizeof(bfloat16_t)];
    __nram__ bfloat16_t src_nram[TOTAL_INPUT_BYTES / sizeof(bfloat16_t)];

    __memcpy(src_nram, src, TOTAL_INPUT_BYTES, GDRAM2NRAM);

    __bang_ssparse_filter_index(dst_nram, src_nram, SRC_CHANNEL_BYTES,
                                KERNEL_HEIGHT, KERNEL_WIDTH, DST_CHANNEL);
    __bang_ssparse_filter_index(
        dst_nram + (INPUT_BYTES / (sizeof(bfloat16_t) * 8 * sizeof(bfloat16_t))),
        src_nram + (INPUT_BYTES / sizeof(bfloat16_t)), SRC_CHANNEL_BYTES, KERNEL_
        HEIGHT,
```

```
    KERNEL_WIDTH, DST_CHANNEL);

__memcpy(dst, dst_nram, TOTAL_OUTPUT_BYTES, NRAM2GDRAM);
}
```

3.6.22 __bang_ssparse_filter_sparse_index

```
void __bang_ssparse_filter_sparse_index(half *dst,
                                         half *src,
                                         half *index,
                                         int input_channel,
                                         int src_height,
                                         int src_width,
                                         int output_channel)
```

```
void __bang_ssparse_filter_sparse_index(float *dst,
                                         float *src,
                                         float *index,
                                         int input_channel,
                                         int src_height,
                                         int src_width,
                                         int output_channel)
```

```
void __bang_ssparse_filter_sparse_index(int8_t *dst,
                                         int8_t *src,
                                         int8_t *index,
                                         int input_channel,
                                         int src_height,
                                         int src_width,
                                         int output_channel)
```

```
void __bang_ssparse_filter_sparse_index(int16_t *dst,
                                         int16_t *src,
                                         int16_t *index,
                                         int input_channel,
                                         int src_height,
                                         int src_width,
                                         int output_channel)
```

```
void __bang_ssparse_filter_sparse_index(int *dst,
                                         int *src,
                                         int *index,
                                         int input_channel,
                                         int src_height,
                                         int src_width,
                                         int output_channel)
```

```
void __bang_ssparse_filter_sparse_index(bfloat16_t *dst,
                                         bfloat16_t *src,
                                         bfloat16_t *index,
                                         int input_channel,
                                         int src_height,
                                         int src_width,
                                         int output_channel)
```

Selects 2 elements based on <index> from every 4 elements of <src>, sets the unselected elements to zero and stores the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] index: The address of index vector.
- [in] input_channel: The number of input channel of source vector.
- [in] src_height: The height of source vector.
- [in] src_width: The width of source vector.
- [in] output_channel: The number of channel of destination vector.

Return

- void

Remark

- <dst>, <src> and <index> must point to __nram__ address space;
- Each bit in <index> corresponds to one element in <src>;
- <dst> and <src> have the same <input_channel> dimension size;
- The byte size of <input_channel> dimension of <index> equals ceil(<input_channel> / bit_size(typeof(<src>)));
- If <index> is split by 4 bits and some parts do not belong to {4'b0011, 4'b0101, 4'b1001, 4'b0110, 4'b1010, 4'b1100}, hardware will set illegal parts to 4'b0011 to keep the program running.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```

#include "bang.h"

#define SRC_CHANNEL_BYTES (2 * 128)
#define KERNEL_HEIGHT 1
#define KERNEL_WIDTH 1
#define DST_CHANNEL 1

#define INPUT_BYTES ((SRC_CHANNEL_BYTES) * (KERNEL_HEIGHT) * (KERNEL_WIDTH) * (DST_
→CHANNEL))
#define OUTPUT_BYTES (INPUT_BYTES)

#define TOTAL_INPUT_BYTES (2 * (INPUT_BYTES))
#define TOTAL_OUTPUT_BYTES (2 * (OUTPUT_BYTES))

__mlu_global__ void kernel(bfloat16_t* dst, bfloat16_t* src) {
    __nram__ bfloat16_t dst_nram[TOTAL_OUTPUT_BYTES / sizeof(bfloat16_t)];
    __nram__ bfloat16_t src_nram[TOTAL_INPUT_BYTES / sizeof(bfloat16_t)];
    __nram__ bfloat16_t idx_nram[TOTAL_INPUT_BYTES / sizeof(bfloat16_t) / 4];

    __memcpy(src_nram, src, TOTAL_INPUT_BYTES, GDRAM2NRAM);

    __bang_ssparse_filter_index(idx_nram,
                                src_nram,
                                SRC_CHANNEL_BYTES,
                                SRC_HEIGHT,
                                SRC_WIDTH,
                                DST_CHANNEL);

    __bang_ssparse_filter_sparse_index(dst_nram,
                                       src_nram,
                                       idx_nram,
                                       SRC_CHANNEL_BYTES,
                                       SRC_HEIGHT,
                                       SRC_WIDTH,
                                       DST_CHANNEL);

    __bang_ssparse_filter_index(
        (bfloat16_t*)((char*)(idx_nram) + INPUT_BYTES / (sizeof(bfloat16_t) * 8)),
        (bfloat16_t*)((char*)(src_nram) + INPUT_BYTES), SRC_CHANNEL_BYTES, KERNEL_
→HEIGHT,
        KERNEL_WIDTH, DST_CHANNEL);
    __bang_ssparse_filter_sparse_index(
        (bfloat16_t*)((char*)(dst_nram) + OUTPUT_BYTES),
        (bfloat16_t*)((char*)(src_nram) + INPUT_BYTES),
        (bfloat16_t*)((char*)(idx_nram) + (INPUT_BYTES / (sizeof(bfloat16_t) * 8))),
        SRC_CHANNEL_BYTES, KERNEL_HEIGHT, KERNEL_WIDTH, DST_CHANNEL);
    __memcpy(dst, dst_nram, TOTAL_OUTPUT_BYTES, NRAM2GDRAM);
}

```

Cambricon@155chb

3.6.23 __bang_ssparse_filter_union

```
void __bang_ssparse_filter_union(half *dst,
                                 half *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_union(float *dst,
                                 float *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_union(int8_t *dst,
                                 int8_t *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_union(int16_t *dst,
                                 int16_t *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_union(int *dst,
                                 int *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

```
void __bang_ssparse_filter_union(bfloat16_t *dst,
                                 bfloat16_t *src,
                                 int input_channel,
                                 int src_height,
                                 int src_width,
                                 int output_channel)
```

Selects 2 elements with larger absolute value from every 4 elements of <src> and stores the selected elements and corresponding indexes in <dst> in form of 96-byte structure. Each

structure consists of 64-byte data and 32-byte index.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] input_channel: The number of input channel of source vector.
- [in] src_height: The height of source vector.
- [in] src_width: The width of source vector.
- [in] output_channel: The number of channel of destination vector.

Return

- void

Remark

- <dst> and <src> must point to __nram__ address space;
- <src> is split by 128 bytes on input_channel dimension. When the split part is less than 128 bytes, hardware will pad to 128 bytes with zero;
- Each bit in index corresponds to 4 bits in <src>. If one bit is 1 in output index, that means its corresponding 4 bits in <src> are selected;
- There is no alignment constraint for parameter <input_channel>, but the size of input_channel dimension of <dst> must align to (ceil(<input_channel> / 128) * 96) byte(s).

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: --BANG_ARCH >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```
#include "bang.h"

#define SRC_CHANNEL_BYTES (2 * 128)
#define KERNEL_HEIGHT 1
#define KERNEL_WIDTH 1
#define DST_CHANNEL 1
#define INPUT_BYTES \
    ((SRC_CHANNEL_BYTES) * (KERNEL_HEIGHT) * (KERNEL_WIDTH) * (DST_CHANNEL))
#define OUTPUT_BYTES ((INPUT_BYTES) / 128 * 96)
#define TOTAL_INPUT_BYTES (2 * (INPUT_BYTES))
#define TOTAL_OUTPUT_BYTES (2 * (OUTPUT_BYTES))

__mlu_global__ void kernel_ssparse_filter_union(bfloat16_t* dst, bfloat16_t* src) {
    __nram__ bfloat16_t dst_nram[TOTAL_OUTPUT_BYTES / sizeof(bfloat16_t)];
    __nram__ bfloat16_t src_nram[TOTAL_INPUT_BYTES / sizeof(bfloat16_t)];
    __memcpy(src_nram, src, TOTAL_INPUT_BYTES, GDRAM2NRAM);
    __bang_ssparse_filter_union(static_cast<bfloat16_t*>(dst_nram),
                                static_cast<bfloat16_t*>(src_nram), SRC_CHANNEL_BYTES,
                                KERNEL_HEIGHT, KERNEL_WIDTH, DST_CHANNEL);
    __bang_ssparse_filter_union(
```

```
    static_cast<bfloat16_t*>(&dst_nram[OUTPUT_BYTES / sizeof(bfloat16_t)]),  
    static_cast<bfloat16_t*>(&src_nram[INPUT_BYTES / sizeof(bfloat16_t)]), SRC_  
    ->CHANNEL_BYTES,  
    KERNEL_HEIGHT, KERNEL_WIDTH, DST_CHANNEL);  
    __memcpy(dst, dst_nram, TOTAL_OUTPUT_BYTES, NRAM2GDRAM);  
}
```

3.6.24 __bang_ssparse_filter_union_index

```
void __bang_ssparse_filter_union_index(half *dst,  
                                      half *src,  
                                      half *index,  
                                      int input_channel,  
                                      int src_height,  
                                      int src_width,  
                                      int output_channel)
```

```
void __bang_ssparse_filter_union_index(float *dst,  
                                       float *src,  
                                       float *index,  
                                       int input_channel,  
                                       int src_height,  
                                       int src_width,  
                                       int output_channel)
```

```
void __bang_ssparse_filter_union_index(int8_t *dst,  
                                       int8_t *src,  
                                       int8_t *index,  
                                       int input_channel,  
                                       int src_height,  
                                       int src_width,  
                                       int output_channel)
```

```
void __bang_ssparse_filter_union_index(int16_t *dst,  
                                       int16_t *src,  
                                       int16_t *index,  
                                       int input_channel,  
                                       int src_height,  
                                       int src_width,  
                                       int output_channel)
```

```
void __bang_ssparse_filter_union_index(int *dst,
                                       int *src,
                                       int *index,
                                       int input_channel,
                                       int src_height,
                                       int src_width,
                                       int output_channel)
```

```
void __bang_ssparse_filter_union_index(bfloat16_t *dst,
                                       bfloat16_t *src,
                                       bfloat16_t *index,
                                       int input_channel,
                                       int src_height,
                                       int src_width,
                                       int output_channel)
```

Selects 2 elements based on <index> from every 4 elements of <src> and stores the selected elements and corresponding indexes in <dst> in form of 96-byte structure. Each structure consists of 64-byte data and 32-byte index.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] index: The address of index vector.
- [in] input_channel: The number of input channel of source vector.
- [in] src_height: The height of source vector.
- [in] src_width: The width of source vector.
- [in] output_channel: The number of channel of destination vector.

Return

- void.

Remark

- <dst>, <src> and <index> must point to `_nram_` address space;
- Each bit in <index> corresponds to one element in <src>;
- Each bit in output index corresponds to 4 bits in <src>. If one bit is 1 in output index, its corresponding 4 bits in <src> are selected;
- There is no alignment constraint for parameter <input_channel>, but the size of `input_channel` dimension of <dst> must align to 96 bytes;
- The data part of <dst> should be 64-byte aligned. Otherwise, hardware will pad to 64 bytes with zero;
- The index part of <dst> should be 32-byte aligned. Otherwise,
 - If `bit_size(<type>)` == 4, hardware will pad to 32 bytes with `4'b0011`.
 - If `bit_size(<type>)` == 8, hardware will pad to 32 bytes with `8'b0000_1111`.
 - If `bit_size(<type>)` == 16, hardware will pad to 32 bytes with `16'b0000_0000_1111_1111`.
 - If `bit_size(<type>)` == 32, hardware will pad to 32 bytes with `32'b0000_0000_0000_1111_1111_1111_1111`;
- Split <index> by 4 bits and set illegal parts, which do not belong to {`4'b0011`, `4'b0101`,

4'b1001, 4'b0110, 4'b1010, 4'b1100}, to 4'b0011.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 4.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

```
#include "bang.h"

#define SRC_CHANNEL_BYTES (2 * 128)
#define KERNEL_HEIGHT 1
#define KERNEL_WIDTH 1
#define DST_CHANNEL 1
#define INPUT_BYTES \
    ((SRC_CHANNEL_BYTES) * (KERNEL_HEIGHT) * (KERNEL_WIDTH) * (DST_CHANNEL))
#define OUTPUT_BYTES ((INPUT_BYTES) / 128 * 96)
#define TOTAL_INPUT_BYTES (2 * (INPUT_BYTES))
#define TOTAL_OUTPUT_BYTES (2 * (OUTPUT_BYTES))

__mlu_global__ void kernel_ssparse_filter_union_index(bfloat16_t* dst, bfloat16_t* src) {
    __nram__ bfloat16_t dst_nram[TOTAL_OUTPUT_BYTES / sizeof(bfloat16_t)];
    __nram__ bfloat16_t src_nram[TOTAL_INPUT_BYTES / sizeof(bfloat16_t)];
    __nram__ bfloat16_t idx_nram[TOTAL_INPUT_BYTES / sizeof(bfloat16_t) / 4];
    __memcpy(src_nram, src, TOTAL_INPUT_BYTES, GDRAM2NRAM);
    __bang_ssparse_filter_index(static_cast<bfloat16_t*>(idx_nram),
        static_cast<bfloat16_t*>(src_nram), SRC_CHANNEL_BYTES,
        KERNEL_HEIGHT, KERNEL_WIDTH, DST_CHANNEL);
    __bang_ssparse_filter_union_index(
        static_cast<bfloat16_t*>(dst_nram), static_cast<bfloat16_t*>(src_nram),
        static_cast<bfloat16_t*>(idx_nram), SRC_CHANNEL_BYTES, KERNEL_HEIGHT, KERNEL_
        WIDTH,
        DST_CHANNEL);
    __bang_ssparse_filter_index(
        (bfloat16_t*)((char*)(idx_nram) + INPUT_BYTES / (sizeof(bfloat16_t) * 8)),
        (bfloat16_t*)((char*)(src_nram) + INPUT_BYTES), SRC_CHANNEL_BYTES, KERNEL_
        HEIGHT,
        KERNEL_WIDTH, DST_CHANNEL);
    __bang_ssparse_filter_union_index(
        (bfloat16_t*)((char*)(dst_nram) + OUTPUT_BYTES),
        (bfloat16_t*)((char*)(src_nram) + INPUT_BYTES),
        (bfloat16_t*)((char*)(idx_nram) + (INPUT_BYTES / (sizeof(bfloat16_t) * 8))),
        SRC_CHANNEL_BYTES, KERNEL_HEIGHT, KERNEL_WIDTH, DST_CHANNEL);
    __memcpy(dst, dst_nram, TOTAL_OUTPUT_BYTES, NRAM2GDRAM);
}
```

3.6.25 __bang_sumpool

```
void __bang_sumpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width)
```

```
void __bang_sumpool(half *dst,
                     half *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_sumpool(bfloat16_t *dst,
                     bfloat16_t *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

```
void __bang_sumpool(float *dst,
                     float *src,
                     int channel,
                     int height,
                     int width,
                     int kernel_height,
                     int kernel_width,
                     int stride_width,
                     int stride_height)
```

Applies sumpooling forward operation on <src> [`<height>, <width>, <channel>`], a three-dimensional tensor, with sliding window [`<kernel_height>, <kernel_width>`] and stride [`<stride_width>, <stride_height>`], and saves the sum in each window. When window is slid in certain direction (H or W direction), if the left elements number doesn't match the window size, these elements will be discarded.

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of input feature map.
- [in] width: The width of input feature map.
- [in] kernel_height: The height of kernel.
- [in] kernel_width: The width of kernel.
- [in] stride_width: Stride of W direction.
- [in] stride_height: Stride of H direction.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <dst> cannot be overlapped with <src>;
- <channel> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than or equal to 1;
- <stride_height> and <stride_width> must be greater than or equal to 1 if specified.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define CHANNELS 64
#define IN_HEIGHT 9
#define IN_WIDTH 9
#define KERNEL_HEIGHT 3
#define KERNEL_WIDTH 3
#define STRIDE_X 1
#define STRIDE_Y 1
#define INPUT_COUNT ((CHANNELS) * (IN_WIDTH) * (IN_HEIGHT))
#define OUTPUT_COUNT \
    ((CHANNELS) * ((IN_HEIGHT - KERNEL_HEIGHT) / STRIDE_Y + 1) * \
     ((IN_WIDTH - KERNEL_WIDTH) / STRIDE_X + 1))

__mlu_entry__ void PoolSumKernel(half* output, half* input) {
    __nram__ half a_tmp[INPUT_COUNT];
    __nram__ half b_tmp[OUTPUT_COUNT];
    __memcpy(a_tmp, input, INPUT_COUNT * sizeof(half), GDRAM2NRAM);
    __bang_sumpool(b_tmp, a_tmp, CHANNELS, IN_HEIGHT, IN_WIDTH,
```

```

        KERNEL_HEIGHT, KERNEL_WIDTH, STRIDE_X, STRIDE_Y);
__memcpy(output, b_tmp, OUTPUT_COUNT * sizeof(half), NRAM2GDRAM);
}

```

3.6.26 __bang_unpool

```

void __bang_unpool(half *dst,
                    half *src,
                    int channel,
                    int height,
                    int width,
                    int kernel_height,
                    int kernel_width,
                    int stride_width,
                    int stride_height,
                    int index)

```

```

void __bang_unpool(bfloat16_t *dst,
                    bfloat16_t *src,
                    int channel,
                    int height,
                    int width,
                    int kernel_height,
                    int kernel_width,
                    int stride_width,
                    int stride_height,
                    int index)

```

```

void __bang_unpool(float *dst,
                    float *src,
                    int channel,
                    int height,
                    int width,
                    int kernel_height,
                    int kernel_width,
                    int stride_width,
                    int stride_height,
                    int index)

```

Applies unpooling backward operation on `<src>`, a tensor. Every element of `<src>` tensor corresponds to a kernel window on `<dst>` tensor operand. The elements of `<src>` tensor is written into `<index>`' th position in that kernel window on `<dst>` tensor, and other positions in the kernel window are written zero.

The Figure [Process of Unpool Operation](#) shows an example with the following parameters, `height = 5`, `width = 5`, `kernel_height = 2`, `kernel_width = 2`, `stride_height = 3`, `stride_width = 3`, `input_height = 2`, `input_width = 2`, and `index = 1`.

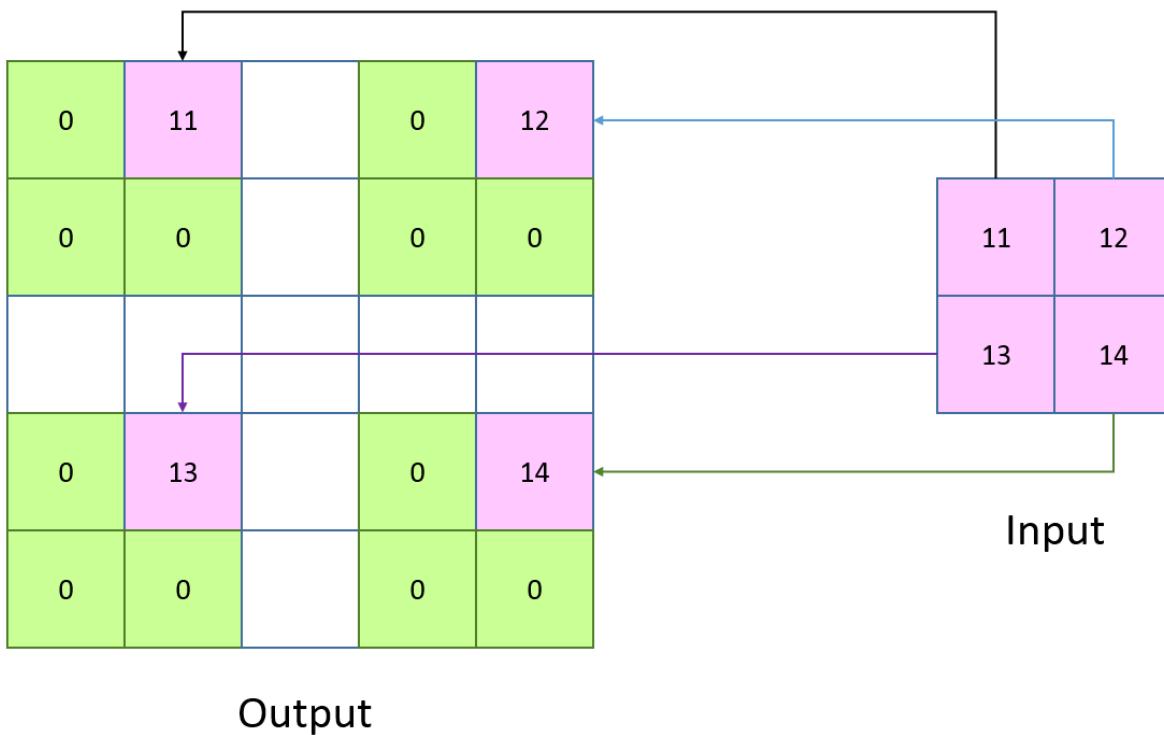


Fig. 3.4: Process of Unpool Operation

Cambricon@155chb

Parameters

- [out] dst: The address of destination tensor whose data layout is HWC.
- [in] src: The address of source tensor whose data layout is HWC.
- [in] channel: Input channel.
- [in] height: The height of output feature map.
- [in] width: The width of output feature map.
- [in] kernel_height: The height of kernel.
- [in] kernel_width: The width of kernel.
- [in] stride_width: W direction of stride.
- [in] stride_height: H direction of stride.
- [in] index: Index within kernel window.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src> and <dst> must point to __nram__ address space;
- <channel> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The shape of <dst> tensor [, , out_channel] and <src> tensor [input_height, input_width, <channel>] has following restrictions:
 $< \text{height} > = (\text{input_height}-1)* < \text{stride_height} > + < \text{kernel_height} >$,
 $< \text{width} > = (\text{input_width}-1)* < \text{stride_width} > + < \text{kernel_width} >$,
 $\text{out_channel} = < \text{channel} >$;

- The <index>' th position in kernel window means ikh' th row and ikw' th column in kernel window, where ($ikh * \text{kernel_width} + ikw = \text{index}$);
- If $\text{stride_width} > \text{kernel_width}$ or $\text{stride_height} > \text{kernel_height}$, the elements in <dst> that do not belong to any kernel window remain unchanged;
- <dst> cannot be overlapped with <src>;
- <channel>, <stride_height>, <stride_width> and <index> must be greater than 0;
- <kernel_height> and <kernel_width> must be greater than 0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ ≥ 200 ;
- CNCC Version: cncc --version $\geq 2.8.0$;
- Cambricon BANG Compute Arch Version: cncc --bang-arch $\geq \text{compute_20}$;
- MLU Compute Arch Version: cncc --bang-mlu-arch $\geq (\text{m})\text{tp_2xx}$.

Example

```
#include <bang.h>

#define CHANNELS 128
#define HEIGHT 6
#define WIDTH 4
#define KERNEL_HEIGHT 3
#define KERNEL_WIDTH 2
#define STRIDE_X 2
#define STRIDE_Y 3
#define INDEX_IN_KERNEL 2
#define H_SMALL ((HEIGHT-KERNEL_HEIGHT)/STRIDE_Y+1)
#define W_SMALL ((WIDTH-KERNEL_WIDTH)/STRIDE_X+1)
#define TOP_DATA_COUNT ((CHANNELS) * (WIDTH) * (HEIGHT))
#define BOTTOM_DATA_COUNT ((CHANNELS) * H_SMALL * W_SMALL)

__mlu_entry__ void UnPoolKernel(half* top_data, int16_t* bottom_data,
                                int channels, int height, int width,
                                int kh, int kw, int sx, int sy,
                                int index_in_kernel) {
    __nram__ half a_tmp[BOTTOM_DATA_COUNT];
    __nram__ half b_tmp[TOP_DATA_COUNT];
    __memcpy(a_tmp, bottom_data, BOTTOM_DATA_COUNT * sizeof(half), GDRAM2NRAM);
    __memcpy(b_tmp, top_data, TOP_DATA_COUNT * sizeof(half), GDRAM2NRAM);
    __bang_unpool(b_tmp, a_tmp, CHANNELS, HEIGHT, WIDTH, KERNEL_HEIGHT,
                  KERNEL_WIDTH, sx, sy, index_in_kernel);
    __memcpy(top_data, b_tmp, TOP_DATA_COUNT * sizeof(half), NRAM2GDRAM);
}
```

3.7 Atomic Functions

3.7.1 __bang_atomic_add

```
void __bang_atomic_add(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_add(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_add(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_add(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_add(half *dst,
                      half *src1,
                      half src2,
                      int size = 1)
```

```
void __bang_atomic_add(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t src2,
                      int size = 1)
```

```
void __bang_atomic_add(float *dst,
                      float *src1,
                      float src2,
                      int size = 1)
```

```
void __bang_atomic_add(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

Cambricon@155chb

```

void __bang_atomic_add(short *dst,
                      short *src1,
                      short *src2,
                      int size)

void __bang_atomic_add(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)

void __bang_atomic_add(int *dst,
                      int *src1,
                      int *src2,
                      int size)

void __bang_atomic_add(half *dst,
                      half *src1,
                      half *src2,
                      int size)

void __bang_atomic_add(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t *src2,
                      int size)

void __bang_atomic_add(float *dst,
                      float *src1,
                      float *src2,
                      int size)

```

Cambricon@155chb

Copies $\langle \text{src1} \rangle$ to $\langle \text{dst} \rangle$, adds $\langle \text{src2} \rangle$ to $\langle \text{src1} \rangle$ element-wisely. That is: $\langle \text{dst} \rangle = \langle \text{src1} \rangle; \langle \text{src1} \rangle = \langle \text{src1} \rangle + \langle \text{src2} \rangle$. All steps are inseparable. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst : The address of destination vector.
- [in] src1 : The address of first source vector.
- [in] src2 : The second source scalar or the address of second source vector.
- [in] size : The elements number of source vector.

Return

- void.

Remark

- $\langle \text{dst} \rangle$ can be overlapped with $\langle \text{src2} \rangle$ if $\langle \text{src2} \rangle$ is a vector;
- bfloat16_t is supported on mtp_592 or higher;
- The address of $\langle \text{src1} \rangle$ must be `sizeof(type)` aligned;
- $\langle \text{dst} \rangle$ must point to `__nram__` address space;
- $\langle \text{src2} \rangle$ must point to `__nram__` address space if $\langle \text{src2} \rangle$ is a vector;

- <size> must be greater than zero;
- <src1> must point to `__mlu_device__` address space.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(short* src1, short src2) {
    __nram__ short v[64];
    __bang_atomic_add(v, src1, src2, 64);
}
```

3.7.2 `__bang_atomic_and`

```
void __bang_atomic_and(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

Cambricon@155chb

```
void __bang_atomic_and(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_and(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_and(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_and(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```

void __bang_atomic_and(short *dst,
                      short *src1,
                      short *src2,
                      int size)

void __bang_atomic_and(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)

void __bang_atomic_and(int *dst,
                      int *src1,
                      int *src2,
                      int size)

```

Applies bitwise AND operation to the vector <src1> and <src2>, stores the result in <src1>, and stores the original value of <src1> in <dst>. That is: <dst> = <src1>; <src1> = <src1> & <src2>. All steps are inseparable.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The address of first source vector.
- [in] src2: The second source scalar or the address of second source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <dst> can be overlapped with <src2> if <src2> is a vector;
- The address of <src1> must be sizeof(type) aligned;
- <src1> must point to __mlu_device__ address space;
- <dst> must point to __nram__ address space;
- <size> must be greater than zero;
- <src2> must point to __nram__ address space if <src2> is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_2xx.

Example

- None.

3.7.3 __bang_atomic_cas

```
void __bang_atomic_cas(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      unsigned short src3)
```

```
void __bang_atomic_cas(short *dst,
                      short *src1,
                      short src2,
                      short src3)
```

```
void __bang_atomic_cas(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      unsigned int src3)
```

```
void __bang_atomic_cas(float *dst,
                      float *src1,
                      float src2,
                      float src3)
```

```
void __bang_atomic_cas(half *dst,
                      half *src1,
                      half src2,
                      half src3)
```

```
void __bang_atomic_cas(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t src2,
                      bfloat16_t src3)
```

```
void __bang_atomic_cas(int *dst,
                      int *src1,
                      int src2,
                      int src3)
```

If `<src2>` is equal to `<*src1>`, stores `<src3>` in `<src1>`. Stores the original value of `<*src1>` in `<dst>`. That is: `<*dst> = <*src1>; <*src1> = (<*src1> == <src2>) ? <src3> : <*src1>`. All steps are inseparable.

Parameters

- [out] `dst`: The address of destination operand.
- [in] `src1`: The address of first operand.
- [in] `src2`: The second operand.
- [in] `src3`: The third operand.

Return

- void.

Remark

- This function only operates on one element;
- <dst> must point to __nram__ address space;
- <src1> must point to __mlu_device__ address space;
- The address of <src1> must be sizeof(type) aligned;
- bfloat16_t, float and half are supported on mtp_592 or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(short* src1, short src2, short src3) {
    __nram__ short v;
    __bang_atomic_cas(&v, src1, src2, src3);
}
```

3.7.4 __bang_atomic_dec

```
void __bang_atomic_dec(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_dec(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_dec(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_dec(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_dec(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```
void __bang_atomic_dec(short *dst,
                      short *src1,
                      short *src2,
                      int size)
```

```
void __bang_atomic_dec(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)
```

```
void __bang_atomic_dec(int *dst,
                      int *src1,
                      int *src2,
                      int size)
```

Compares vector `<src1>` and `<src2>` element-wisely. If `<src1>` is larger than `<src2>`, or the value of `<src1>` is 0, stores the int value `<src2>` in `<src1>`; otherwise, subtracts `<src1>` by 1. Stores the original value of `<src1>` in `<dst>`. That is: $\langle dst \rangle = \langle src1 \rangle; \langle src1 \rangle = (\langle src1 \rangle == 0 || \langle src1 \rangle > \langle src2 \rangle) ? \langle src2 \rangle : (\langle src1 \rangle - 1)$. All steps are inseparable.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The address of first source vector.
- [in] `src2`: The second source scalar or the address of second source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src2>` if `<src2>` is a vector;
- `<size>` must be greater than zero;
- The address of `<src1>` must be `sizeof(type)` aligned;
- `<src1>` must point to `__mlu_device__` address space;
- `<dst>` must point to `__nram__` address space;
- `<src2>` must point to `__nram__` address space if `<src2>` is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx`.

Example

- None.

3.7.5 __bang_atomic_exch

```
void __bang_atomic_exch(unsigned short *dst,
                        unsigned short *src1,
                        unsigned short src2,
                        int size = 1)
```

```
void __bang_atomic_exch(short *dst,
                        short *src1,
                        short src2,
                        int size = 1)
```

```
void __bang_atomic_exch(unsigned int *dst,
                        unsigned int *src1,
                        unsigned int src2,
                        int size = 1)
```

```
void __bang_atomic_exch(int *dst,
                        int *src1,
                        int src2,
                        int size = 1)
```

```
void __bang_atomic_exch(half *dst,
                        half *src1,
                        half src2,
                        int size = 1)
```

```
void __bang_atomic_exch(bfloat16_t *dst,
                        bfloat16_t *src1,
                        bfloat16_t src2,
                        int size = 1)
```

```
void __bang_atomic_exch(float *dst,
                        float *src1,
                        float src2,
                        int size = 1)
```

```
void __bang_atomic_exch(unsigned short *dst,
                        unsigned short *src1,
                        unsigned short *src2,
                        int size)
```

```

void __bang_atomic_exch(short *dst,
                        short *src1,
                        short *src2,
                        int size)

void __bang_atomic_exch(unsigned int *dst,
                        unsigned int *src1,
                        unsigned int *src2,
                        int size)

void __bang_atomic_exch(int *dst,
                        int *src1,
                        int *src2,
                        int size)

void __bang_atomic_exch(half *dst,
                        half *src1,
                        half *src2,
                        int size)

void __bang_atomic_exch(bfloat16_t *dst,
                        bfloat16_t *src1,
                        bfloat16_t *src2,
                        int size)

void __bang_atomic_exch(float *dst,
                        float *src1,
                        float *src2,
                        int size)

```

Stores vector <src1> in <dst>. Stores <src2> in <src1>. That is: <dst> = <src1>; <src1> = <src2>. All steps are inseparable.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The address of first source vector.
- [in] src2: The second source scalar or the address of second source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <dst> can be overlapped with <src2> if <src2> is a vector;
- bfloat16_t is supported on mtp_592 or higher;
- <size> must be greater than zero;
- The address of <src1> must be sizeof(type) aligned;
- <src1> must point to __mlu_device__ address space;
- <dst> must point to __nram__ address space;

- <src2> must point to `__nram__` address space if <src2> is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx.`

Example

- None.

3.7.6 `__bang_atomic_inc`

```
void __bang_atomic_inc(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_inc(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_inc(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_inc(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_inc(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```
void __bang_atomic_inc(short *dst,
                      short *src1,
                      short *src2,
                      int size)
```

```
void __bang_atomic_inc(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)
```

```
void __bang_atomic_inc(int *dst,
                      int *src1,
                      int *src2,
                      int size)
```

Compares vector `<src1>` and `<src2>` element-wisely. If `<src1>` is smaller than `<src2>`, increases `<src1>` by 1; otherwise, sets `<src1>` to 0. Stores the original value of `<src1>` in `<dst>`. That is: `<dst> = <src1>; <src1> = (<src1> >= <src2>) ? 0 : (<src1> + 1)`. All steps are inseparable.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The address of first source vector.
- [in] `src2`: The address of second source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src2>` if `<src2>` is a vector;
- The address of `<src1>` must be `sizeof(type)` aligned;
- `<src1>` must point to `__mlu_device__` address space;
- `<size>` must be greater than zero;
- `<dst>` must point to `__nram__` address space;
- `<src2>` must point to `__nram__` address space if `<src2>` is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx`.

Example

- None.

3.7.7 __bang_atomic_max

```
void __bang_atomic_max(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_max(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_max(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_max(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_max(half *dst,
                      half *src1,
                      half src2,
                      int size = 1)
```

```
void __bang_atomic_max(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t src2,
                      int size = 1)
```

```
void __bang_atomic_max(float *dst,
                      float *src1,
                      float src2,
                      int size = 1)
```

```
void __bang_atomic_max(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```
void __bang_atomic_max(short *dst,
                      short *src1,
                      short *src2,
                      int size)
```

```
void __bang_atomic_max(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)
```

```
void __bang_atomic_max(int *dst,
                      int *src1,
                      int *src2,
                      int size)
```

```
void __bang_atomic_max(half *dst,
                      half *src1,
                      half *src2,
                      int size)
```

```
void __bang_atomic_max(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t *src2,
                      int size)
```

```
void __bang_atomic_max(float *dst,
                      float *src1,
                      float *src2,
                      int size)
```

element-wisely stores the larger value of vector <src1> and <src2> in <src1>. Stores the original value of <src1> in <dst>. That is: <dst> = <src1>; <src1> = (<src1> > <src2>) ? <src1> : <src2>. All steps are inseparable. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The address of first source vector.
- [in] src2: The address of second source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- bfloat16_t is supported on mtp_592 or higher;
- <src1> must point to __mlu_device__ address space;
- <dst> can be overlapped with <src2> if <src2> is a vector;
- The address of <src1> must be sizeof(type) aligned;
- <size> must be greater than zero;
- <dst> must point to __nram__ address space;
- <src2> must point to __nram__ address space if <src2> is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_2xx.

Example

- None.

3.7.8 __bang_atomic_min

```
void __bang_atomic_min(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_min(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_min(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_min(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_min(half *dst,
                      half *src1,
                      half src2,
                      int size = 1)
```

```
void __bang_atomic_min(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t src2,
                      int size = 1)
```

```
void __bang_atomic_min(float *dst,
                      float *src1,
                      float src2,
                      int size = 1)
```

```
void __bang_atomic_min(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```
void __bang_atomic_min(short *dst,
                      short *src1,
                      short *src2,
                      int size)
```

```
void __bang_atomic_min(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)
```

```
void __bang_atomic_min(int *dst,
                      int *src1,
                      int *src2,
                      int size)
```

```
void __bang_atomic_min(half *dst,
                      half *src1,
                      half *src2,
                      int size)
```

```
void __bang_atomic_min(bfloat16_t *dst,
                      bfloat16_t *src1,
                      bfloat16_t *src2,
                      int size)
```

```
void __bang_atomic_min(float *dst,
                      float *src1,
                      float *src2,
                      int size)
```

Takes the smaller value from two vector values `<src1>` and `<src2>`, and stores it in `<src1>`. Stores the original value of `<src1>` in `<dst>`. That is: `<dst> = <src1>`; `<src1> = (<src1> < <src2>) ? <src1> : <src2>`. All steps are inseparable. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The address of first source vector.
- [in] `src2`: The second source scalar or the address of second source vector.
- [in] `size`: The elements number of source vector.

Return

- void.

Remark

- <src1> must point to `__mlu_device__` address space;
- <dst> can be overlapped with <src2> if <src2> is a vector;
- The address of <src1> must be `sizeof(type)` aligned;
- `bfloat16_t` is supported on `mtp_592` or higher;
- <size> must be greater than zero;
- <dst> must point to `__nram__` address space;
- <src2> must point to `__nram__` address space if <src2> is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx`.

Example

- None.

3.7.9 `__bang_atomic_or`

```
void __bang_atomic_or(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_or(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_or(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_or(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_or(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```

void __bang_atomic_or(short *dst,
                      short *src1,
                      short *src2,
                      int size)

void __bang_atomic_or(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)

void __bang_atomic_or(int *dst,
                      int *src1,
                      int *src2,
                      int size)

```

Applies bitwise OR operation to vector <src1> and <src2>, stores the result in <src1>, and stores the original value of <src1> in <dst>. That is: <dst> = <src1>; <src1> = <src1> | <src2>. All steps are inseparable.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The address of first source vector.
- [in] src2: The second source scalar or the address of second source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <dst> must point to __nram__ address space;
- <src1> must point to __mlu_device__ address space;
- <dst> can be overlapped with <src2> if <src2> is a vector;
- The address of <src1> must be sizeof(type) aligned;
- <size> must be greater than zero;
- <src2> must point to __nram__ address space if <src2> is a vector.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_2xx.

Example

- None.

3.7.10 __bang_atomic_xor

```
void __bang_atomic_xor(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short src2,
                      int size = 1)
```

```
void __bang_atomic_xor(short *dst,
                      short *src1,
                      short src2,
                      int size = 1)
```

```
void __bang_atomic_xor(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int src2,
                      int size = 1)
```

```
void __bang_atomic_xor(int *dst,
                      int *src1,
                      int src2,
                      int size = 1)
```

```
void __bang_atomic_xor(unsigned short *dst,
                      unsigned short *src1,
                      unsigned short *src2,
                      int size)
```

```
void __bang_atomic_xor(short *dst,
                      short *src1,
                      short *src2,
                      int size)
```

```
void __bang_atomic_xor(unsigned int *dst,
                      unsigned int *src1,
                      unsigned int *src2,
                      int size)
```

```
void __bang_atomic_xor(int *dst,
                      int *src1,
                      int *src2,
                      int size)
```

Applies bitwise XOR operation to vector <src1> and <src2>, stores the result in <src1>, and stores the original value of <src1> in <dst>. That is: <dst> = <src1>; <src1> = <src1> ^ <src2>. All steps are inseparable.

Parameters

- [out] dst: The address of destination vector.

- [in] `src1`: The address of first source vector.
- [in] `src2`: The second source scalar or the address of second source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<dst>` must point to `__nram__` address space;
- `<src2>` must point to `__nram__` address space if `<src2>` is a vector;
- `<src1>` must point to `__mlu_device__` address space;
- `<size>` must be greater than zero;
- `<dst>` can be overlapped with `<src2>` if `<src2>` is a vector;
- The address of `<src1>` must be `sizeof(type)` aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx`.

Example

- None.

3.8 Atomic Reduce Functions

3.8.1 __bang_atomic_reduce_add

```
void __bang_atomic_reduce_add(short *dst,
                             short src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_add(int *dst,
                             int src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_add(half *dst,
                             half src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_add(bfloat16_t *dst,
                             bfloat16_t src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_add(float *dst,
                             float src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_add(short *dst,
                             short *src1,
                             int size)
```

```
void __bang_atomic_reduce_add(int *dst,
                             int *src1,
                             int size)
```

```
void __bang_atomic_reduce_add(half *dst,
                             half *src1,
                             int size)
```

```
void __bang_atomic_reduce_add(bfloat16_t *dst,
                             bfloat16_t *src1,
                             int size)
```

```
void __bang_atomic_reduce_add(float *dst,
                             float *src1,
                             int size)
```

Increases `<dst>` by `<src1>` element-wisely. That is: `<dst> = <dst> + <src1>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Cambricon@155chb

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The first source scalar or the address of first source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<src1>` must point to `__nram__` address space if `<src1>` is a vector;
- `<dst>` must point to `__mlu_device__` address space;
- The address of `<dst>` must be `sizeof(type)` aligned;
- `<size>` must be greater than zero;
- `bfloat16_t` is supported on `mtp_592` or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372`.

Example

```
#include <bang.h>
```

```
--mlu_entry__ void kernel(short* src1, short src2, int size) {
    __bang_atomic_reduce_add(src1, src2, size);
}
```

3.8.2 __bang_atomic_reduce_and

```
void __bang_atomic_reduce_and(short *dst,
                             short src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_and(int *dst,
                             int src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_and(short *dst,
                             short *src1,
                             int size)
```

```
void __bang_atomic_reduce_and(int *dst,
                             int *src1,
                             int size)
```

Applies bitwise AND operation to the vector <dst> and <src1>, and stores the result in <dst>. That is: <dst> = <dst> & <src1>.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The first source scalar or the address of first source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <src1> must point to __nram__ address space if <src1> is a vector;
- <dst> must point to __mlu_device__ address space;
- <size> must be greater than zero;
- The address of <dst> must be sizeof(type) aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 372;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_372.

Example

- None.

3.8.3 __bang_atomic_reduce_cas

```
void __bang_atomic_reduce_cas(short *dst,
                             short src1,
                             short src2)
```

```
void __bang_atomic_reduce_cas(half *dst,
                             half src1,
                             half src2)
```

```
void __bang_atomic_reduce_cas(float *dst,
                             float src1,
                             float src2)
```

```
void __bang_atomic_reduce_cas(bfloat16_t *dst,
                             bfloat16_t src1,
                             bfloat16_t src2)
```

```
void __bang_atomic_reduce_cas(int *dst,
                             int src1,
                             int src2)
```

If `<src1>` is equal to `<*dst>`, stores `<src2>` in `<dst>`. That is: `<*dst> = (<*dst> == <src1>) ? <src2> : <*dst>`.

Cambricon@155chb

Parameters

- [out] `dst`: The address of destination operand.
- [in] `src1`: The first operand.
- [in] `src2`: The second operand.

Return

- `void`.

Remark

- This function only operates on one element;
- `bfloat16_t` is supported on `mtp_592` or higher;
- `<dst>` must point to `__mlu_device__` address space;
- The address of `<dst>` must be `sizeof(type)` aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(int* dst, int src1, int src2) {
```

```

    __bang_atomic_reduce_cas(dst, src1, src2);
}

```

3.8.4 [__bang_atomic_reduce_dec](#)

```
void __bang_atomic_reduce_dec(unsigned short *dst,
                             unsigned short src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_dec(short *dst,
                             short src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_dec(unsigned int *dst,
                             unsigned int src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_dec(int *dst,
                             int src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_dec(unsigned short *dst,
                             unsigned short *src1,
                             int size)
```

```
void __bang_atomic_reduce_dec(short *dst,
                             short *src1,
                             int size)
```

```
void __bang_atomic_reduce_dec(unsigned int *dst,
                             unsigned int *src1,
                             int size)
```

```
void __bang_atomic_reduce_dec(int *dst,
                             int *src1,
                             int size)
```

Compares vector <dst> and <src1>. If <dst> is larger than <src1>, or the value of <dst> is 0, stores the int value <src1> in <dst>; otherwise, reduces <dst> by 1. That is: $<\text{dst}> = (\text{<dst>} == 0 \parallel \text{<dst>} > \text{<src1>}) ? \text{<src1>} : (\text{<dst>} - 1)$.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The first source scalar or the address of first source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <src1> must point to `__nram__` address space if <src1> is a vector;
- <dst> must point to `__mlu_device__` address space;
- <size> must be greater than zero;
- The address of <dst> must be `sizeof(type)` aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372.`

Example

- None.

3.8.5 `__bang_atomic_reduce_exch`

```
void __bang_atomic_reduce_exch(short *dst,
                               short src1,
                               int size = 1)
```

```
void __bang_atomic_reduce_exch(int *dst,
                               int src1,
                               int size = 1)
```

```
void __bang_atomic_reduce_exch(half *dst,
                               half src1,
                               int size = 1)
```

```
void __bang_atomic_reduce_exch(bfloat16_t *dst,
                               bfloat16_t src1,
                               int size = 1)
```

```
void __bang_atomic_reduce_exch(float *dst,
                               float src1,
                               int size = 1)
```

```
void __bang_atomic_reduce_exch(short *dst,
                               short *src1,
                               int size)
```

```
void __bang_atomic_reduce_exch(int *dst,
                               int *src1,
                               int size)
```

```
void __bang_atomic_reduce_exch(half *dst,
                               half *src1,
                               int size)
```

```
void __bang_atomic_reduce_exch(bfloat16_t *dst,
                               bfloat16_t *src1,
                               int size)
```

```
void __bang_atomic_reduce_exch(float *dst,
                               float *src1,
                               int size)
```

Stores <src1> in <dst>. That is: <dst> = <src1>.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The first source scalar or the address of first source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <src1> must point to __nram__ address space if <src1> is a vector;
- <dst> must point to __mlu_device__ address space;
- <size> must be greater than zero;
- The address of <dst> must be sizeof(type) aligned;
- bfloat16_t is supported on mtp_592 or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 372;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_372.

Example

- None.

3.8.6 __bang_atomic_reduce_inc

```
void __bang_atomic_reduce_inc(unsigned short *dst,
                             unsigned short src1,
                             int size = 1)
```

```
void __bang_atomic_reduce_inc(short *dst,
                             short src1,
                             int size = 1)
```

```

void __bang_atomic_reduce_inc(unsigned int *dst,
                             unsigned int src1,
                             int size = 1)

void __bang_atomic_reduce_inc(int *dst,
                             int src1,
                             int size = 1)

void __bang_atomic_reduce_inc(unsigned short *dst,
                             unsigned short *src1,
                             int size)

void __bang_atomic_reduce_inc(short *dst,
                             short *src1,
                             int size)

void __bang_atomic_reduce_inc(unsigned int *dst,
                             unsigned int *src1,
                             int size)

```

Cambricon@155chb

Compares vector `<dst>` and `<src1>`. If `<dst>` is smaller than `<src1>`, increases `<dst>` by 1; otherwise, sets `<dst>` to 0. That is: $<\text{dst}> = (\text{<dst>} \geq \text{<src1>}) ? 0 : (\text{<dst>} + 1)$.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The first source scalar or the address of first source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<src1>` must point to `__nram__` address space if `<src1>` is a vector;
- `<dst>` must point to `__mlu_device__` address space;
- `<size>` must be greater than zero;
- The address of `<dst>` must be `sizeof(type)` aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372`.

Example

- None.

3.8.7 __bang_atomic_reduce_max

```
void __bang_atomic_reduce_max(unsigned short *dst,  
                             unsigned short src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(short *dst,  
                             short src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(unsigned int *dst,  
                             unsigned int src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(int *dst,  
                             int src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(half *dst,  
                             half src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(bfloat16_t *dst,  
                             bfloat16_t src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(float *dst,  
                             float src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_max(unsigned short *dst,  
                             unsigned short *src1,  
                             int size)
```

```
void __bang_atomic_reduce_max(short *dst,  
                             short *src1,  
                             int size)
```

```
void __bang_atomic_reduce_max(unsigned int *dst,  
                             unsigned int *src1,  
                             int size)
```

```
void __bang_atomic_reduce_max(int *dst,  
                             int *src1,  
                             int size)
```

```
void __bang_atomic_reduce_max(half *dst,
                             half *src1,
                             int size)
```

```
void __bang_atomic_reduce_max(bfloat16_t *dst,
                             bfloat16_t *src1,
                             int size)
```

```
void __bang_atomic_reduce_max(float *dst,
                             float *src1,
                             int size)
```

Takes the larger value from vector `<dst>` and `<src1>`, and stores it in `<dst>`. That is: `<dst> = (<dst> > <src1>) ? <dst> : <src1>`. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The first source scalar or the address of first source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<src1>` must point to `__nram__` address space if `<src1>` is a vector;
- `<dst>` must point to `__mlu_device__` address space;
- The address of `<dst>` must be `sizeof(type)` aligned;
- `<size>` must be greater than zero;
- `bfloat16_t` is supported on `mtp_592` or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372`.

Example

- None.

3.8.8 __bang_atomic_reduce_min

```
void __bang_atomic_reduce_min(unsigned short *dst,
                            unsigned short src1,
                            int size = 1)
```

```
void __bang_atomic_reduce_min(short *dst,  
                             short src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_min(unsigned int *dst,  
                             unsigned int src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_min(int *dst,  
                             int src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_min(half *dst,  
                             half src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_min(bfloat16_t *dst,  
                             bfloat16_t src1,  
                             int size = 1)
```

```
void __bang_atomic_reduce_min(float *dst,  
                             float src1,  
                             int size = 1)
```

Cambricon@155chb

```
void __bang_atomic_reduce_min(unsigned short *dst,  
                             unsigned short *src1,  
                             int size)
```

```
void __bang_atomic_reduce_min(short *dst,  
                             short *src1,  
                             int size)
```

```
void __bang_atomic_reduce_min(unsigned int *dst,  
                             unsigned int *src1,  
                             int size)
```

```
void __bang_atomic_reduce_min(int *dst,  
                             int *src1,  
                             int size)
```

```
void __bang_atomic_reduce_min(half *dst,  
                             half *src1,  
                             int size)
```

```
void __bang_atomic_reduce_min(bfloat16_t *dst,
                             bfloat16_t *src1,
                             int size)
```

```
void __bang_atomic_reduce_min(float *dst,
                             float *src1,
                             int size)
```

Takes the smaller value from two vector values `<dst>` and `<src1>`, and stores it in `<dst>`. That is: `<dst> = (<dst> < <src1>) ? <dst> : <src1>`. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src1`: The first source scalar or the address of first source vector.
- [in] `size`: The elements number of source vector.

Return

- `void`.

Remark

- `<src1>` must point to `__nram__` address space if `<src1>` is a vector;
- `<dst>` must point to `__mlu_device__` address space;
- The address of `<src1>` must be `sizeof(type)` aligned;
- `<size>` must be greater than zero;
- `bfloat16_t` is supported on `mtp_592` or higher.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372`.

Example

- None.

3.8.9 __bang_atomic_reduce_or

```
void __bang_atomic_reduce_or(short *dst,
                            short src1,
                            int size = 1)
```

```
void __bang_atomic_reduce_or(int *dst,
                            int src1,
                            int size = 1)
```

```
void __bang_atomic_reduce_or(short *dst,
                            short *src1,
                            int size)
```

```
void __bang_atomic_reduce_or(int *dst,
                            int *src1,
                            int size)
```

Applies bitwise OR operation to vector <dst> and <src1>, and stores the result in <dst>. That is: <dst> = <dst> | <src1>.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The first source scalar or the address of first source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <src1> must point to `__nram__` address space if <src1> is a vector;
- <dst> must point to `__mlu_device__` address space;
- <size> must be greater than zero;
- The address of <dst> must be `sizeof(type)` aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 372`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_372`.

Example

- None.

3.8.10 `__bang_atomic_reduce_xor`

```
void __bang_atomic_reduce_xor(short *dst,
                            short src1,
                            int size = 1)
```

```
void __bang_atomic_reduce_xor(int *dst,
                            int src1,
                            int size = 1)
```

```
void __bang_atomic_reduce_xor(short *dst,
                            short *src1,
                            int size)
```

```
void __bang_atomic_reduce_xor(int *dst,
                             int *src1,
                             int size)
```

Applies bitwise XOR operation to vector <dst> and <src1>, and stores the result in <dst>. That is: <dst> = <dst> ^ <src1>.

Parameters

- [out] dst: The address of destination vector.
- [in] src1: The first source scalar or the address of first source vector.
- [in] size: The elements number of source vector.

Return

- void.

Remark

- <src1> must point to __nram__ address space if <src1> is a vector;
- <dst> must point to __mlu_device__ address space;
- <size> must be greater than zero;
- The address of <dst> must be sizeof(type) aligned.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 372;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_372.

Example

- None.

3.9 Control Flow and Debugging Functions

3.9.1 __assert

```
void __assert(const char *__assertion,
             const char *__file,
             int __line)
```

Makes assertion in the code, prints assert message and abort. __assert is the same as that of in C language.

<assert.h> should be included. For example, void assert(int exp), when exp is 0, message will be printed as: Cambricon BANG Assertion failed: expression, file filename, line nnn. If macro NDEBUG is defined, macro assert will be ignored.

Parameters

- [in] __assertion: assertion message.
- [in] __file: assertion file.
- [in] __line: assertion line number.

Return

- void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.9.2 __bang_printf

```
int __bang_printf(const char *fmt,
                  ...)
```

Similar to standard printf function, prints arguments to screen, formatted by the format string.

Parameters

- [in] fmt: The format string, which must be a literal constant string.

Return

- void.

Cambricon@155chb

Remark

- <fmt> must be a literal constant string;
- This function can at most accept 17 parameters including the format string;
- The type of parameters can be different. It must be one of: char, unsigned char, short, unsigned short, int, unsigned int, half, float, char, bool, pointer;
- Since this function involves extremely time-consuming interaction between CPU and MLU, it might cause significant performance degradation.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.9.3 __is_ipu

```
bool __is_ipu()
```

Returns true for MLU cores.

Return

- bool.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.4.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.9.4 __is_mpu

```
bool __is_mpu()
```

Returns true for MPU cores.

Cambricon@155chb

Return

- bool.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.4.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.9.5 __is_nram

```
bool __is_nram(const void *ptr)
```

Returns true if <ptr> points to __nram__ address space.

Parameters

- [in] ptr: The input pointer.

Return

- bool.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.4.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.9.6 __is_sram

```
bool __is_sram(const void *ptr)
```

Returns true if <ptr> points to __mlu_shared__ address space.

Parameters

- [in] ptr: The input pointer.

Return

- bool.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.4.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.9.7 __is_wram

```
bool __is_wram(const void *ptr)
```

Returns true if <ptr> points to __wram__ address space.

Parameters

- [in] ptr: The input pointer.

Return

- bool.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.4.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.10 Discrete Atomic Functions

Cambricon@155chb

3.10.1 __bang_discrete_atomic_add

```
void __bang_discrete_atomic_add(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(half *dst,
                                half *src1_base,
                                char *src1_offset,
                                half *src2,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_add(half *dst,
                                half *src1_base,
                                short *src1_offset,
                                half *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(float *dst,
                                float *src1_base,
                                char *src1_offset,
                                float *src2,
                                int size)
```

```
void __bang_discrete_atomic_add(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 int size)
```

```
void __bang_discrete_atomic_add(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 int size)
```

```
void __bang_discrete_atomic_add(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 char *src1_offset,
                                 bfloat16_t *src2,
                                 int size)
```

```
void __bang_discrete_atomic_add(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 short *src1_offset,
                                 bfloat16_t *src2,
                                 int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_add(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 int *src1_offset,
                                 bfloat16_t *src2,
                                 int size)
```

```
void __bang_discrete_atomic_add(short *dst,
                                 short *src1_base,
                                 char *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(short *dst,
                                 short *src1_base,
                                 short *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_add(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(int *dst,
                                 int *src1_base,
                                 int *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(half *dst,
                                 half *src1_base,
                                 char *src1_offset,
                                 half *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(half *dst,
                                 half *src1_base,
                                 short *src1_offset,
                                 half *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_add(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 char *src1_offset,
                                 bfloat16_t *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 short *src1_offset,
                                 bfloat16_t *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_add(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Adds <src2> to the values in discrete <src1> addresses. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- The address of <src1_base> must be sizeof(type) aligned;
- <src1_base> must point to __mlu_device__ address space;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic add is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(short* acc, char* offset, short* src2, int* mask) {
    __nram__ short v[128];
    __nram__ char offset_nram[128];
    __nram__ short src2_nram[128];
    __nram__ int mask_nram[128 / sizeof(int) / 8];
    __memcpy(offset_nram, offset, 128 * sizeof(char), GDRAM2NRAM);
    __memcpy(src2_nram, src2, 128 * sizeof(short), GDRAM2NRAM);
    __memcpy(mask_nram, mask, 128 / 8, GDRAM2NRAM);
```

```
    __bang_discrete_atomic_add(v, acc, offset_nram, src2_nram, mask_nram, 128);  
}
```

3.10.2 __bang_discrete_atomic_and

```
void __bang_discrete_atomic_and(short *dst,  
                                short *src1_base,  
                                char *src1_offset,  
                                short *src2,  
                                int size)
```

```
void __bang_discrete_atomic_and(short *dst,  
                                short *src1_base,  
                                short *src1_offset,  
                                short *src2,  
                                int size)
```

```
void __bang_discrete_atomic_and(short *dst,  
                                short *src1_base,  
                                int *src1_offset,  
                                short *src2,  
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_and(int *dst,  
                                int *src1_base,  
                                char *src1_offset,  
                                int *src2,  
                                int size)
```

```
void __bang_discrete_atomic_and(int *dst,  
                                int *src1_base,  
                                short *src1_offset,  
                                int *src2,  
                                int size)
```

```
void __bang_discrete_atomic_and(int *dst,  
                                int *src1_base,  
                                int *src1_offset,  
                                int *src2,  
                                int size)
```

```
void __bang_discrete_atomic_and(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_and(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_and(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_and(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_and(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_and(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Performs bitwise AND on the values in discrete <src1> addresses with <src2> respectively. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- <src1_base> must point to __mlu_device__ address space;
- The address of <src1_base> must be sizeof(type) aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic bitwise AND is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.10.3 __bang_discrete_atomic_cas

```
void __bang_discrete_atomic_cas(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                short src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                short src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                short src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 int src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 int src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(int *dst,
                                 int *src1_base,
                                 int *src1_offset,
                                 int *src2,
                                 int src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(half *dst,
                                 half *src1_base,
                                 char *src1_offset,
                                 half *src2,
                                 half src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(half *dst,
                                 half *src1_base,
                                 short *src1_offset,
                                 half *src2,
                                 half src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                half src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 float src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 float src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 float src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 char *src1_offset,
                                 bfloat16_t *src2,
                                 bfloat16_t src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 short *src1_offset,
                                 bfloat16_t *src2,
                                 bfloat16_t src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                bfloat16_t src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(short *dst,
                                 short *src1_base,
                                 char *src1_offset,
                                 short *src2,
                                 void *mask,
                                 short src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(short *dst,
                                 short *src1_base,
                                 short *src1_offset,
                                 short *src2,
                                 void *mask,
                                 short src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(short *dst,
                                 short *src1_base,
                                 int *src1_offset,
                                 short *src2,
                                 void *mask,
                                 short src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                void *mask,
                                int src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                void *mask,
                                int src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(half *dst,
                                half *src1_base,
                                char *src1_offset,
                                half *src2,
                                void *mask,
                                half src3,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_cas(half *dst,
                                half *src1_base,
                                short *src1_offset,
                                half *src2,
                                void *mask,
                                half src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                void *mask,
                                half src3,
                                int size)
```

```
void __bang_discrete_atomic_cas(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 void *mask,
                                 float src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 void *mask,
                                 float src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 void *mask,
                                 float src3,
                                 int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_cas(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 char *src1_offset,
                                 bfloat16_t *src2,
                                 void *mask,
                                 bfloat16_t src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(bfloat16_t *dst,
                                 bfloat16_t *src1_base,
                                 short *src1_offset,
                                 bfloat16_t *src2,
                                 void *mask,
                                 bfloat16_t src3,
                                 int size)
```

```
void __bang_discrete_atomic_cas(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                bfloat16_t src3,
                                int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Stores the values in <src2> in discrete <src1> addresses respectively if they are equal to <src3>. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] src3: The third operand.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

Cambricon@155chb

- <size> must be greater than zero;
- <src1_base> must point to __mlu_device__ address space;
- The address of <src1_base> must be sizeof(type) aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic compare-and-swap is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.10.4 __bang_discrete_atomic_dec

```
void __bang_discrete_atomic_dec(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned short *dst,
                                unsigned short *src1_base,
                                char *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned short *dst,
                                unsigned short *src1_base,
                                short *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned int *dst,
                                unsigned int *src1_base,
                                char *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned int *dst,
                                unsigned int *src1_base,
                                short *src1_offset,
                                unsigned int *src2,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_dec(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_dec(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned short *dst,
                                unsigned short *src1_base,
                                char *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned short *dst,
                                unsigned short *src1_base,
                                short *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_dec(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```

void __bang_discrete_atomic_dec(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                void *mask,
                                int size)

void __bang_discrete_atomic_dec(unsigned int *dst,
                                 unsigned int *src1_base,
                                 char *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)

void __bang_discrete_atomic_dec(unsigned int *dst,
                                 unsigned int *src1_base,
                                 short *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)

void __bang_discrete_atomic_dec(unsigned int *dst,
                                 unsigned int *src1_base,
                                 int *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)

```

Computes the discrete `<src1>` addresses by adding offsets stored in `<src1_offset>` to `<src1_base>`. Decrements the values in discrete `<src1>` addresses by 1 if they are not greater than the corresponding values in `<src2>` and not equal to 0; otherwise, sets them to the corresponding values in `<src2>`. Stores the original values in discrete `<src1>` addresses in `<dst>`. All steps are inseparable.

Parameters

- [out] `dst`: The address of destination operand.
- [in] `src1_base`: The base address of the first operand.
- [in] `src1_offset`: The address of the offsets of the first operand.
- [in] `src2`: The second operand.
- [in] `mask`: The address of mask.
- [in] `size`: The number of elements to be computed.

Return

- `void`.

Remark

- `<size>` must be greater than zero;
- `<src1_base>` must point to `__mlu_device__` address space;
- The address of `<src1_base>` must be `sizeof(type)` aligned;

- <dst>, <src1_offset>, <src2> and <mask> must point to `_nram_` address space;
- The offsets stored in <src1_offset> are in bytes, and must be `sizeof(type)` aligned;
- <mask> has <size> effective bits, and each bit controls if atomic decrement is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

- None.

3.10.5 `__bang_discrete_atomic_exch`

```
void __bang_discrete_atomic_exch(short *dst,
                                  short *src1_base,
                                  char *src1_offset,
                                  short *src2,
                                  int size)
```

```
void __bang_discrete_atomic_exch(short *dst,
                                  short *src1_base,
                                  short *src1_offset,
                                  short *src2,
                                  int size)
```

```
void __bang_discrete_atomic_exch(short *dst,
                                  short *src1_base,
                                  int *src1_offset,
                                  short *src2,
                                  int size)
```

```
void __bang_discrete_atomic_exch(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 int size)
```

```
void __bang_discrete_atomic_exch(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 int size)
```

```
void __bang_discrete_atomic_exch(int *dst,
                                  int *src1_base,
                                  int *src1_offset,
                                  int *src2,
                                  int size)

void __bang_discrete_atomic_exch(half *dst,
                                  half *src1_base,
                                  char *src1_offset,
                                  half *src2,
                                  int size)

void __bang_discrete_atomic_exch(half *dst,
                                  half *src1_base,
                                  short *src1_offset,
                                  half *src2,
                                  int size)

void __bang_discrete_atomic_exch(half *dst,
                                  half *src1_base,
                                  int *src1_offset,
                                  half *src2,
                                  int size)

void __bang_discrete_atomic_exch(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 int size)

void __bang_discrete_atomic_exch(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 int size)

void __bang_discrete_atomic_exch(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_exch(bfloat16_t *dst,
                                  bfloat16_t *src1_base,
                                  char *src1_offset,
                                  bfloat16_t *src2,
                                  int size)
```

```
void __bang_discrete_atomic_exch(bfloat16_t *dst,
                                  bfloat16_t *src1_base,
                                  short *src1_offset,
                                  bfloat16_t *src2,
                                  int size)
```

```
void __bang_discrete_atomic_exch(bfloat16_t *dst,
                                  bfloat16_t *src1_base,
                                  int *src1_offset,
                                  bfloat16_t *src2,
                                  int size)
```

```
void __bang_discrete_atomic_exch(short *dst,
                                  short *src1_base,
                                  char *src1_offset,
                                  short *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(short *dst,
                                  short *src1_base,
                                  short *src1_offset,
                                  short *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(short *dst,
                                  short *src1_base,
                                  int *src1_offset,
                                  short *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(int *dst,
                                  int *src1_base,
                                  char *src1_offset,
                                  int *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(int *dst,
                                  int *src1_base,
                                  short *src1_offset,
                                  int *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(int *dst,
                                  int *src1_base,
                                  int *src1_offset,
                                  int *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(half *dst,
                                  half *src1_base,
                                  char *src1_offset,
                                  half *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(half *dst,
                                  half *src1_base,
                                  short *src1_offset,
                                  half *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(half *dst,
                                  half *src1_base,
                                  int *src1_offset,
                                  half *src2,
                                  void *mask,
                                  int size)
```

```
void __bang_discrete_atomic_exch(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 void *mask,
                                 int size)
```

```

void __bang_discrete_atomic_exch(float *dst,
                                  float *src1_base,
                                  short *src1_offset,
                                  float *src2,
                                  void *mask,
                                  int size)

void __bang_discrete_atomic_exch(float *dst,
                                  float *src1_base,
                                  int *src1_offset,
                                  float *src2,
                                  void *mask,
                                  int size)

void __bang_discrete_atomic_exch(bfloat16_t *dst,
                                  bfloat16_t *src1_base,
                                  char *src1_offset,
                                  bfloat16_t *src2,
                                  void *mask,
                                  int size)

void __bang_discrete_atomic_exch(bfloat16_t *dst,
                                  bfloat16_t *src1_base,
                                  short *src1_offset,
                                  bfloat16_t *src2,
                                  void *mask,
                                  int size)

void __bang_discrete_atomic_exch(bfloat16_t *dst,
                                  bfloat16_t *src1_base,
                                  int *src1_offset,
                                  bfloat16_t *src2,
                                  void *mask,
                                  int size)

```

Computes the discrete `<src1>` addresses by adding offsets stored in `<src1_offset>` to `<src1_base>`. Stores the values in `<src2>` in discrete `<src1>` addresses respectively. Stores the original values in discrete `<src1>` addresses in `<dst>`. All steps are inseparable.

Parameters

- [out] `dst`: The address of destination operand.
- [in] `src1_base`: The base address of the first operand.
- [in] `src1_offset`: The address of the offsets of first operand.
- [in] `src2`: The second operand.
- [in] `mask`: The address of mask.
- [in] `size`: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- <src1_base> must point to __mlu_device__ address space;
- The address of <src1_base> must be sizeof(type) aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic exchange is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.10.6 __bang_discrete_atomic_inc

```
void __bang_discrete_atomic_inc(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(unsigned short *dst,
                                unsigned short *src1_base,
                                char *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(unsigned short *dst,
                                unsigned short *src1_base,
                                short *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_inc(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(unsigned int *dst,
                                unsigned int *src1_base,
                                char *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(unsigned int *dst,
                                unsigned int *src1_base,
                                short *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_inc(short *dst,
                                 short *src1_base,
                                 char *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(short *dst,
                                 short *src1_base,
                                 short *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(short *dst,
                                 short *src1_base,
                                 int *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(unsigned short *dst,
                                 unsigned short *src1_base,
                                 char *src1_offset,
                                 unsigned short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(unsigned short *dst,
                                 unsigned short *src1_base,
                                 short *src1_offset,
                                 unsigned short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_inc(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(int *dst,
                                 int *src1_base,
                                 int *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(unsigned int *dst,
                                 unsigned int *src1_base,
                                 char *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(unsigned int *dst,
                                 unsigned int *src1_base,
                                 short *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_inc(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                void *mask,
                                int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Increments the values in discrete <src1> addresses by 1 if they are smaller than the corresponding values in <src2>; otherwise, sets them to 0. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- <src1_base> must point to __mlu_device__ address space;
- The address of <src1_base> must be sizeof(type) aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic increment is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.10.7 __bang_discrete_atomic_max

```
void __bang_discrete_atomic_max(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned short *dst,
                                unsigned short *src1_base,
                                char *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned short *dst,
                                unsigned short *src1_base,
                                short *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_max(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                int size)

void __bang_discrete_atomic_max(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                int size)

void __bang_discrete_atomic_max(unsigned int *dst,
                                unsigned int *src1_base,
                                char *src1_offset,
                                unsigned int *src2,
                                int size)

void __bang_discrete_atomic_max(unsigned int *dst,
                                unsigned int *src1_base,
                                short *src1_offset,
                                unsigned int *src2,
                                int size)

void __bang_discrete_atomic_max(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                int size)

void __bang_discrete_atomic_max(half *dst,
                                half *src1_base,
                                char *src1_offset,
                                half *src2,
                                int size)

void __bang_discrete_atomic_max(half *dst,
                                half *src1_base,
                                short *src1_offset,
                                half *src2,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_max(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                int size)

void __bang_discrete_atomic_max(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 int size)

void __bang_discrete_atomic_max(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 int size)

void __bang_discrete_atomic_max(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 int size)

void __bang_discrete_atomic_max(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                char *src1_offset,
                                bfloat16_t *src2,
                                int size)

void __bang_discrete_atomic_max(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                short *src1_offset,
                                bfloat16_t *src2,
                                int size)

void __bang_discrete_atomic_max(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_max(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned short *dst,
                                unsigned short *src1_base,
                                char *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned short *dst,
                                unsigned short *src1_base,
                                short *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned int *dst,
                                unsigned int *src1_base,
                                char *src1_offset,
                                unsigned int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned int *dst,
                                unsigned int *src1_base,
                                short *src1_offset,
                                unsigned int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(half *dst,
                                half *src1_base,
                                char *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(half *dst,
                                half *src1_base,
                                short *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(float *dst,
                                float *src1_base,
                                char *src1_offset,
                                float *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(float *dst,
                                float *src1_base,
                                short *src1_offset,
                                float *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(float *dst,
                                float *src1_base,
                                int *src1_offset,
                                float *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                char *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                short *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_max(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Sets the values in discrete <src1> addresses to the greater values between the original values and the corresponding values in <src2>. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- <src1_base> must point to `__mlu_device__` address space;
- The address of <src1_base> must be `sizeof(type)` aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to `__nram__` address space;
- The offsets stored in <src1_offset> are in bytes, and must be `sizeof(type)` aligned;
- <mask> has <size> effective bits, and each bit controls if atomic maximum is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.10.8 __bang_discrete_atomic_min

```
void __bang_discrete_atomic_min(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(unsigned short *dst,
                                unsigned short *src1_base,
                                char *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(unsigned short *dst,
                                unsigned short *src1_base,
                                short *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(unsigned int *dst,
                                unsigned int *src1_base,
                                char *src1_offset,
                                unsigned int *src2,
                                int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_min(unsigned int *dst,
                                unsigned int *src1_base,
                                short *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(half *dst,
                                half *src1_base,
                                char *src1_offset,
                                half *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(half *dst,
                                half *src1_base,
                                short *src1_offset,
                                half *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(float *dst,
                                 float *src1_base,
                                 char *src1_offset,
                                 float *src2,
                                 int size)
```

```
void __bang_discrete_atomic_min(float *dst,
                                 float *src1_base,
                                 short *src1_offset,
                                 float *src2,
                                 int size)
```

Cambricon@155chb

```
void __bang_discrete_atomic_min(float *dst,
                                 float *src1_base,
                                 int *src1_offset,
                                 float *src2,
                                 int size)
```

```
void __bang_discrete_atomic_min(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                char *src1_offset,
                                bfloat16_t *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                short *src1_offset,
                                bfloat16_t *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                int size)
```

```
void __bang_discrete_atomic_min(short *dst,
                                 short *src1_base,
                                 char *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(short *dst,
                                 short *src1_base,
                                 short *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(short *dst,
                                 short *src1_base,
                                 int *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(unsigned short *dst,
                                 unsigned short *src1_base,
                                 char *src1_offset,
                                 unsigned short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(unsigned short *dst,
                                 unsigned short *src1_base,
                                 short *src1_offset,
                                 unsigned short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(unsigned short *dst,
                                unsigned short *src1_base,
                                int *src1_offset,
                                unsigned short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_min(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(int *dst,
                                 int *src1_base,
                                 int *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(unsigned int *dst,
                                 unsigned int *src1_base,
                                 char *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(unsigned int *dst,
                                 unsigned int *src1_base,
                                 short *src1_offset,
                                 unsigned int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_min(unsigned int *dst,
                                unsigned int *src1_base,
                                int *src1_offset,
                                unsigned int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_min(half *dst,
                                half *src1_base,
                                char *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_min(half *dst,
                                half *src1_base,
                                short *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_min(half *dst,
                                half *src1_base,
                                int *src1_offset,
                                half *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_min(float *dst,
                                float *src1_base,
                                char *src1_offset,
                                float *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_min(float *dst,
                                float *src1_base,
                                short *src1_offset,
                                float *src2,
                                void *mask,
                                int size)
```

```

void __bang_discrete_atomic_min(float *dst,
                                float *src1_base,
                                int *src1_offset,
                                float *src2,
                                void *mask,
                                int size)

void __bang_discrete_atomic_min(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                char *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)

void __bang_discrete_atomic_min(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                short *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)

void __bang_discrete_atomic_min(bfloat16_t *dst,
                                bfloat16_t *src1_base,
                                int *src1_offset,
                                bfloat16_t *src2,
                                void *mask,
                                int size)

```

Computes the discrete `<src1>` addresses by adding offsets stored in `<src1_offset>` to `<src1_base>`. Sets the values in discrete `<src1>` addresses to the smaller values between the original values and the corresponding values in `<src2>`. Stores the original values in discrete `<src1>` addresses in `<dst>`. All steps are inseparable. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination operand.
- [in] `src1_base`: The base address of the first operand.
- [in] `src1_offset`: The address of the offsets of the first operand.
- [in] `src2`: The second operand.
- [in] `mask`: The address of mask.
- [in] `size`: The number of elements to be computed.

Return

- `void`.

Remark

- `<size>` must be greater than zero;
- `<src1_base>` must point to `__mlu_device__` address space;

- The address of <src1_base> must be `sizeof(type)` aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to `__nram__` address space;
- The offsets stored in <src1_offset> are in bytes, and must be `sizeof(type)` aligned;
- <mask> has <size> effective bits, and each bit controls if atomic minimum is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

- None.

3.10.9 `__bang_discrete_atomic_or`

```
void __bang_discrete_atomic_or(short *dst,
                               short *src1_base,
                               char *src1_offset,
                               short *src2,
                               int size)
```

```
void __bang_discrete_atomic_or(short *dst,
                               short *src1_base,
                               short *src1_offset,
                               short *src2,
                               int size)
```

```
void __bang_discrete_atomic_or(short *dst,
                               short *src1_base,
                               int *src1_offset,
                               short *src2,
                               int size)
```

```
void __bang_discrete_atomic_or(int *dst,
                               int *src1_base,
                               char *src1_offset,
                               int *src2,
                               int size)
```

```
void __bang_discrete_atomic_or(int *dst,
                               int *src1_base,
                               short *src1_offset,
                               int *src2,
                               int size)
```

```
void __bang_discrete_atomic_or(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                int size)
```

```
void __bang_discrete_atomic_or(short *dst,
                                short *src1_base,
                                char *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_or(short *dst,
                                short *src1_base,
                                short *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_or(short *dst,
                                short *src1_base,
                                int *src1_offset,
                                short *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_or(int *dst,
                                int *src1_base,
                                char *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_or(int *dst,
                                int *src1_base,
                                short *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

```
void __bang_discrete_atomic_or(int *dst,
                                int *src1_base,
                                int *src1_offset,
                                int *src2,
                                void *mask,
                                int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Performs bitwise OR on the values in discrete <src1> addresses with <src2> respectively. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- <src1_base> must point to __mlu_device__ address space;
- The address of <src1_base> must be sizeof(type) aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic bitwise OR is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.10.10 __bang_discrete_atomic_xor

```
void __bang_discrete_atomic_xor(short *dst,
                                 short *src1_base,
                                 char *src1_offset,
                                 short *src2,
                                 int size)
```

```
void __bang_discrete_atomic_xor(short *dst,
                                 short *src1_base,
                                 short *src1_offset,
                                 short *src2,
                                 int size)
```

```
void __bang_discrete_atomic_xor(short *dst,
                                 short *src1_base,
                                 int *src1_offset,
                                 short *src2,
                                 int size)
```

```
void __bang_discrete_atomic_xor(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 int size)
```

```
void __bang_discrete_atomic_xor(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 int size)
```

```
void __bang_discrete_atomic_xor(int *dst,
                                 int *src1_base,
                                 int *src1_offset,
                                 int *src2,
                                 int size)
```

```
void __bang_discrete_atomic_xor(short *dst,
                                 short *src1_base,
                                 char *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_xor(short *dst,
                                 short *src1_base,
                                 short *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_xor(short *dst,
                                 short *src1_base,
                                 int *src1_offset,
                                 short *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_xor(int *dst,
                                 int *src1_base,
                                 char *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_xor(int *dst,
                                 int *src1_base,
                                 short *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

```
void __bang_discrete_atomic_xor(int *dst,
                                 int *src1_base,
                                 int *src1_offset,
                                 int *src2,
                                 void *mask,
                                 int size)
```

Computes the discrete <src1> addresses by adding offsets stored in <src1_offset> to <src1_base>. Performs bitwise XOR on the values in discrete <src1> addresses with <src2> respectively. Stores the original values in discrete <src1> addresses in <dst>. All steps are inseparable.

Parameters

- [out] dst: The address of destination operand.
- [in] src1_base: The base address of the first operand.
- [in] src1_offset: The address of the offsets of the first operand.
- [in] src2: The second operand.
- [in] mask: The address of mask.
- [in] size: The number of elements to be computed.

Return

- void.

Remark

- <size> must be greater than zero;
- <src1_base> must point to __mlu_device__ address space;
- The address of <src1_base> must be sizeof(type) aligned;
- <dst>, <src1_offset>, <src2> and <mask> must point to __nram__ address space;
- The offsets stored in <src1_offset> are in bytes, and must be sizeof(type) aligned;
- <mask> has <size> effective bits, and each bit controls if atomic bitwise XOR is performed on the corresponding element. If no <mask> is given, all elements are computed.

Instruction Pipeline

- IO.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- None.

3.11 Scalar Type Conversion Functions

Cambricon@155chb

3.11.1 __bfloat162char

```
char __bfloat162char(bfloat16_t a)
```

This function converts type of a from bfloat16_t to char in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(bfloat16_t a) {
    char result;
    result = __bfloat162char(a);
}
```

3.11.2 __bfloat162char_dn

```
char __bfloat162char_dn(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `char` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

Cambricon@155chb

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of `__bfloat162char` for more details.

3.11.3 __bfloat162char_oz

```
char __bfloat162char_oz(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `char` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162char](#) for more details.

3.11.4 [__bfloat162char_rd](#)

```
char __bfloat162char_rd(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `char` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162char](#) for more details.

3.11.5 [__bfloat162char_rm](#)

```
char __bfloat162char_rm(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `char` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162char](#) for more details.

3.11.6 [__bfloat162char_rn](#)

```
char __bfloat162char_rn(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `char` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162char](#) for more details.

3.11.7 [__bfloat162char_tz](#)

```
char __bfloat162char_tz(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162char](#) for more details.

3.11.8 [__bfloat162char_up](#)

```
char __bfloat162char_up(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `char` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162char](#) for more details.

3.11.9 [__bfloat162float](#)

```
float __bfloat162float(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(bfloat16_t a) {
    float result;
    result = __bfloat162float(a);
}
```

3.11.10 __bfloat162float_dn

```
float __bfloat162float_dn(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `float` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of `__bfloat162float` for more details.

3.11.11 __bfloat162float_oz

```
float __bfloat162float_oz(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `float` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162float](#) for more details.

3.11.12 [__bfloat162float_rd](#)

```
float __bfloat162float_rd(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `float` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162float](#) for more details.

3.11.13 [__bfloat162float_rm](#)

```
float __bfloat162float_rm(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `float` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162float](#) for more details.

3.11.14 [__bfloat162float_rn](#)

```
float __bfloat162float_rn(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `float` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162float](#) for more details.

3.11.15 [__bfloat162float_tz](#)

```
float __bfloat162float_tz(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162float](#) for more details.

3.11.16 [__bfloat162float_up](#)

```
float __bfloat162float_up(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `float` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162float](#) for more details.

3.11.17 [__bfloat162half](#)

```
half __bfloat162half(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(bfloat16_t a) {
    half result;
    result = __bfloat162half(a);
}
```

3.11.18 __bfloat162half_dn

half __bfloat162half_dn(bfloat16_t a)

This function converts type of a from bfloat16_t to half in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162half](#) for more details.

3.11.19 __bf16half_oz

```
half __bf16half_oz(bf16_t a)
```

This function converts type of `a` from `bf16_t` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16half](#) for more details.

3.11.20 __bf16half_rd

```
half __bf16half_rd(bf16_t a)
```

This function converts type of `a` from `bf16_t` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16half](#) for more details.

3.11.21 `__bf16half_rm`

```
half __bf16half_rm(bf16_t a)
```

This function converts type of `a` from `bf16_t` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16half](#) for more details.

3.11.22 `__bf16half_rn`

```
half __bf16half_rn(bf16_t a)
```

This function converts type of `a` from `bf16_t` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16half](#) for more details.

3.11.23 `__bf16half_tz`

```
half __bf16half_tz(bf16_t a)
```

This function converts type of `a` from `bf16_t` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16half](#) for more details.

3.11.24 `__bf16half_up`

```
half __bf16half_up(bf16_t a)
```

This function converts type of `a` from `bf16_t` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16half](#) for more details.

3.11.25 __bf16int

```
int __bf16int(bf16_t a)
```

This function converts type of a from `bf16_t` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>
__mlu_entry__ void kernel(bf16_t a) {
    int result;
    result = __bf16int(a);
}
```

3.11.26 __bf16int_dn

```
int __bf16int_dn(bf16_t a)
```

This function converts type of a from `bf16_t` to `int` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bfloating16int](#) for more details.

3.11.27 __bfloating16int_oz

```
int __bfloating16int_oz(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `int` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bfloating16int](#) for more details.

3.11.28 __bfloating16int_rd

```
int __bfloating16int_rd(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `int` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bfloating16int](#) for more details.

3.11.29 __bfloating16int_rm

```
int __bfloating16int_rm(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `int` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bfloating16int](#) for more details.

3.11.30 __bfloating16int_rn

```
int __bfloating16int_rn(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `int` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bfloating16int](#) for more details.

3.11.31 __bfloating16int_tz

```
int __bfloating16int_tz(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bfloating16int](#) for more details.

3.11.32 __bfloating16int_up

```
int __bfloating16int_up(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `int` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of `__bfloating162int` for more details.

3.11.33 `__bfloating162short`

```
short __bfloating162short(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(bfloat16_t a) {
    short result;
    result = __bfloating162short(a);
}
```

3.11.34 `__bfloating162short_dn`

```
short __bfloating162short_dn(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `short` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.35 __bfloat162short_oz

```
short __bfloat162short_oz(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `short` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.36 __bfloat162short_rd

```
short __bfloat162short_rd(bfloat16_t a)
```

This function converts type of `a` from `bfloat16_t` to `short` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.37 __bfloat162short_rm

```
short __bfloat162short_rm(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `short` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.38 __bfloat162short_rn

```
short __bfloat162short_rn(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `short` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.39 __bfloat162short_tz

```
short __bfloat162short_tz(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.40 __bfloat162short_up

```
short __bfloat162short_up(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `short` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__bfloat162short](#) for more details.

3.11.41 __bfloat162tf32

```
float __bfloat162tf32(bfloat16_t a)
```

This function converts type of a from `bfloat16_t` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(bfloat16_t a) {
    float result;
    result = __bfloat162tf32(a);
}
```

3.11.42 __bf16_t tf32_dn

```
float __bf16_t tf32_dn(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bf16_t tf32](#) for more details.

3.11.43 __bf16_t tf32_oz

```
float __bf16_t tf32_oz(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bf16_t tf32](#) for more details.

3.11.44 __bf16_t tf32_rd

```
float __bf16_t tf32_rd(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16_t tf32](#) for more details.

3.11.45 __bf16_t tf32_rm

```
float __bf16_t tf32_rm(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__bf16_t tf32](#) for more details.

3.11.46 __bf16_t tf32_rn

```
float __bf16_t tf32_rn(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bf16_t tf32](#) for more details.

3.11.47 __bf16_t tf32_tz

```
float __bf16_t tf32_tz(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bf16_t tf32](#) for more details.

3.11.48 __bf16_t**float162tf32_up**

```
float __bf16_tfloat162tf32_up(bf16_t a)
```

This function converts type of a from `bf16_t` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__bf16_t**float162tf32**](#) for more details.

3.11.49 __char**2bf16**

```
bf16_t __char2bf16(char a)
```

This function converts type of a from `char` to `bf16_t`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(char a) {
    bfloat16_t result;
    result = __char2bfloat16(a);
}
```

3.11.50 __char2float

```
float __char2float(char a)
```

This function converts type of `a` from `char` to `float`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH-- >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.11.51 __char2half

```
half __char2half(char a)
```

This function converts type of `a` from `char` to `half`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH-- >= 200;`

- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.11.52 __char2tf32

```
float __char2tf32(char a)
```

This function converts type of `a` from `char` to `tf32`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(char a) {
    float result;
    result = __char2tf32(a);
}
```

3.11.53 __float2bfloat16

```
bfloat16_t __float2bfloat16(float a)
```

This function converts type of `a` from `float` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    bfloat16_t result;
    result = __float2bfloat16(a);
}
```

3.11.54 __float2bfloat16_dn

```
bfloat16_t __float2bfloat16_dn(float a)
```

This function converts type of a from float to bfloat16_t in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.55 __float2bfloat16_oz

```
bfloat16_t __float2bfloat16_oz(float a)
```

This function converts type of a from float to bfloat16_t in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.56 __float2bfloat16_rd

```
bfloat16_t __float2bfloat16_rd(float a)
```

This function converts type of a from float to bfloat16_t in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.57 __float2bfloat16_rm

```
bfloat16_t __float2bfloat16_rm(float a)
```

This function converts type of a from float to bfloat16_t in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.58 __float2bfloat16_rn

```
bfloat16_t __float2bfloat16_rn(float a)
```

This function converts type of a from float to bfloat16_t in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.59 __float2bfloat16_tz

```
bfloat16_t __float2bfloat16_tz(float a)
```

This function converts type of a from float to bfloat16_t in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.60 __float2bfloat16_up

```
bfloat16_t __float2bfloat16_up(float a)
```

This function converts type of a from float to bfloat16_t in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2bfloat16](#) for more details.

3.11.61 __float2char

```
char __float2char(float a)
```

This function converts type of a from `float` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>
```

```
__mlu_entry__ void kernel(float a) {
    char result;
    result = __float2char(a);
}
```

3.11.62 __float2char_dn

```
char __float2char_dn(float a)
```

This function converts type of a from `float` to `char` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.8.0`;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__float2char` for more details.

3.11.63 __float2char_oz

```
char __float2char_oz(float a)
```

This function converts type of `a` from `float` to `char` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__float2char` for more details.

3.11.64 __float2char_rd

```
char __float2char_rd(float a)
```

This function converts type of `a` from `float` to `char` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2char](#) for more details.

3.11.65 __float2char_rm

```
char __float2char_rm(float a)
```

This function converts type of `a` from `float` to `char` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__float2char](#) for more details.

3.11.66 __float2char_rn

```
char __float2char_rn(float a)
```

This function converts type of `a` from `float` to `char` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__float2char](#) for more details.

3.11.67 __float2char_tz

```
char __float2char_tz(float a)
```

This function converts type of `a` from `float` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark**Instruction Pipeline**

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2char](#) for more details.

3.11.68 __float2char_up

```
char __float2char_up(float a)
```

This function converts type of `a` from `float` to `char` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2char](#) for more details.

3.11.69 __float2half

```
half __float2half(float a)
```

This function converts type of `a` from `float` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2half_tz](#) for more details.

3.11.70 __float2half_dn

```
half __float2half_dn(float a)
```

This function converts type of `a` from `float` to `half` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__float2half_tz` for more details.

3.11.71 __float2half_oz

```
half __float2half_oz(float a)
```

This function converts type of `a` from `float` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__float2half_tz` for more details.

3.11.72 __float2half_rd

```
half __float2half_rd(float a)
```

This function converts type of `a` from `float` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__float2half_tz` for more details.

3.11.73 __float2half_rm

```
half __float2half_rm(float a)
```

This function converts type of `a` from `float` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__float2half_tz` for more details.

3.11.74 __float2half_rn

```
half __float2half_rn(float a)
```

This function converts type of `a` from `float` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__float2half_tz` for more details.

3.11.75 __float2half_tz

```
half __float2half_tz(float a)
```

This function converts type of `a` from `float` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    half result;
    result = __float2half_tz(a);
}
```

3.11.76 __float2half_up

```
half __float2half_up(float a)
```

This function converts type of `a` from `float` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2half_tz](#) for more details.

3.11.77 __float2int

```
int __float2int(float a)
```

This function converts type of `a` from `float` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2int_tz](#) for more details.

3.11.78 __float2int_dn

```
int __float2int_dn(float a)
```

This function converts type of `a` from `float` to `int` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2int_tz](#) for more details.

3.11.79 __float2int_oz

```
int __float2int_oz(float a)
```

This function converts type of `a` from `float` to `int` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2int_tz](#) for more details.

3.11.80 __float2int_rd

```
int __float2int_rd(float a)
```

This function converts type of `a` from `float` to `int` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2int_tz](#) for more details.

3.11.81 __float2int_rm

```
int __float2int_rm(float a)
```

This function converts type of a from `float` to `int` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__float2int_tz](#) for more details.

3.11.82 __float2int_rn

```
int __float2int_rn(float a)
```

This function converts type of a from `float` to `int` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__float2int_tz` for more details.

3.11.83 __float2int_tz

```
int __float2int_tz(float a)
```

This function converts type of `a` from `float` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    int result;
    result = __float2int_tz(a);
}
```

3.11.84 __float2int_up

```
int __float2int_up(float a)
```

This function converts type of `a` from `float` to `int` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2int_tz](#) for more details.

3.11.85 __float2short

Cambricon@155chb

```
short __float2short(float a)
```

This function converts type of `a` from `float` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2short_tz](#) for more details.

3.11.86 __float2short_dn

```
short __float2short_dn(float a)
```

This function converts type of a from `float` to `short` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__float2short_tz` for more details.

3.11.87 __float2short_oz

```
short __float2short_oz(float a)
```

This function converts type of a from `float` to `short` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__float2short_tz` for more details.

3.11.88 __float2short_rd

```
short __float2short_rd(float a)
```

This function converts type of a from `float` to `short` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2short_tz](#) for more details.

3.11.89 __float2short_rm

```
short __float2short_rm(float a)
```

This function converts type of a from `float` to `short` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__float2short_tz](#) for more details.

3.11.90 __float2short_rn

```
short __float2short_rn(float a)
```

This function converts type of a from `float` to `short` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__float2short_tz` for more details.

3.11.91 __float2short_tz

```
short __float2short_tz(float a)
```

This function converts type of a from `float` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    short result;
    result = __float2short_tz(a);
}
```

3.11.92 __float2short_up

```
short __float2short_up(float a)
```

This function converts type of a from float to short in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

Cambricon@155chb

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2short_tz](#) for more details.

3.11.93 __float2tf32

```
float __float2tf32(float a)
```

This function converts type of a from float to tf32 in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    float result;
    result = __float2tf32(a);
}
```

3.11.94 __float2tf32_dn

```
float __float2tf32_dn(float a)
```

This function converts type of a from float to tf32 in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.95 __float2tf32_oz

```
float __float2tf32_oz(float a)
```

This function converts type of a from float to tf32 in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.96 __float2tf32_rd

```
float __float2tf32_rd(float a)
```

This function converts type of a from float to tf32 in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.97 __float2tf32_rm

```
float __float2tf32_rm(float a)
```

This function converts type of a from float to tf32 in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.98 __float2tf32_rn

```
float __float2tf32_rn(float a)
```

This function converts type of a from float to tf32 in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.99 __float2tf32_tz

```
float __float2tf32_tz(float a)
```

This function converts type of a from float to tf32 in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.100 __float2tf32_up

```
float __float2tf32_up(float a)
```

This function converts type of a from `float` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__float2tf32](#) for more details.

3.11.101 __float2uchar

```
unsigned char __float2uchar(float a)
```

This function converts type of a from `float` to `unsigned char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    unsigned char result;
    result = __float2uchar(a);
}
```

3.11.102 __float2uchar_dn

```
unsigned char __float2uchar_dn(float a)
```

This function converts type of a from float to unsigned char in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.103 __float2uchar_oz

```
unsigned char __float2uchar_oz(float a)
```

This function converts type of a from float to unsigned char in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.104 __float2uchar_rd

Cambricon@155chb

```
unsigned char __float2uchar_rd(float a)
```

This function converts type of a from float to unsigned char in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.105 __float2uchar_rm

```
unsigned char __float2uchar_rm(float a)
```

This function converts type of a from float to unsigned char in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.106 __float2uchar_rn

Cambricon@155chb

```
unsigned char __float2uchar_rn(float a)
```

This function converts type of a from float to unsigned char in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.107 __float2uchar_tz

```
unsigned char __float2uchar_tz(float a)
```

This function converts type of a from float to unsigned char in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.108 __float2uchar_up

Cambricon@155chb

```
unsigned char __float2uchar_up(float a)
```

This function converts type of a from float to unsigned char in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2uchar](#) for more details.

3.11.109 __float2uint

```
unsigned int __float2uint(float a)
```

This function converts type of a from `float` to `unsigned int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>
__mlu_entry__ void kernel(float a) {
    unsigned int result;
    result = __float2uint(a);
}
```

3.11.110 __float2uint_dn

```
unsigned int __float2uint_dn(float a)
```

This function converts type of a from `float` to `unsigned int` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2uint](#) for more details.

3.11.111 __float2uint_oz

```
unsigned int __float2uint_oz(float a)
```

This function converts type of `a` from `float` to `unsigned int` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2uint](#) for more details.

3.11.112 __float2uint_rd

```
unsigned int __float2uint_rd(float a)
```

This function converts type of `a` from `float` to `unsigned int` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__float2uint` for more details.

3.11.113 __float2uint_rm

```
unsigned int __float2uint_rm(float a)
```

This function converts type of `a` from `float` to `unsigned int` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__float2uint` for more details.

3.11.114 __float2uint_rn

```
unsigned int __float2uint_rn(float a)
```

This function converts type of `a` from `float` to `unsigned int` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__float2uint](#) for more details.

3.11.115 __float2uint_tz

```
unsigned int __float2uint_tz(float a)
```

This function converts type of a from `float` to `unsigned int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2uint](#) for more details.

3.11.116 __float2uint_up

```
unsigned int __float2uint_up(float a)
```

This function converts type of a from `float` to `unsigned int` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2uint](#) for more details.

3.11.117 __float2ushort

```
unsigned short __float2ushort(float a)
```

This function converts type of `a` from `float` to `unsigned short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    unsigned short result;
    result = __float2ushort(a);
}
```

3.11.118 __float2ushort_dn

```
unsigned short __float2ushort_dn(float a)
```

This function converts type of `a` from `float` to `unsigned short` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2ushort](#) for more details.

3.11.119 __float2ushort_oz

```
unsigned short __float2ushort_oz(float a)
```

This function converts type of a from `float` to `unsigned short` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2ushort](#) for more details.

3.11.120 __float2ushort_rd

```
unsigned short __float2ushort_rd(float a)
```

This function converts type of a from `float` to `unsigned short` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2ushort](#) for more details.

3.11.121 __float2ushort_rm

```
unsigned short __float2ushort_rm(float a)
```

This function converts type of `a` from `float` to `unsigned short` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__float2ushort](#) for more details.

3.11.122 __float2ushort_rn

```
unsigned short __float2ushort_rn(float a)
```

This function converts type of `a` from `float` to `unsigned short` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__float2ushort](#) for more details.

3.11.123 __float2ushort_tz

```
unsigned short __float2ushort_tz(float a)
```

This function converts type of `a` from `float` to `unsigned short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__float2ushort](#) for more details.

3.11.124 __float2ushort_up

```
unsigned short __float2ushort_up(float a)
```

This function converts type of `a` from `float` to `unsigned short` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__float2ushort](#) for more details.

3.11.125 [__half2bfloat16](#)

```
bfloat16_t __half2bfloat16(half a)
```

This function converts type of a from half to bfloat16_t in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    bfloat16_t result;
    result = __half2bfloat16(a);
}
```

3.11.126 __half2bfloat16_dn

```
bfloat16_t __half2bfloat16_dn(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.127 __half2bfloat16_oz

```
bfloat16_t __half2bfloat16_oz(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.128 __half2bfloat16_rd

```
bfloat16_t __half2bfloat16_rd(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.129 __half2bfloat16_rm

```
bfloat16_t __half2bfloat16_rm(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.130 __half2bfloat16_rn

```
bfloat16_t __half2bfloat16_rn(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.131 __half2bfloat16_tz

```
bfloat16_t __half2bfloat16_tz(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.132 __half2bfloat16_up

```
bfloat16_t __half2bfloat16_up(half a)
```

This function converts type of a from `half` to `bfloat16_t` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__half2bfloat16](#) for more details.

3.11.133 __half2char

```
char __half2char(half a)
```

This function converts type of a from `half` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    char result;
    result = __half2char(a);
}
```

3.11.134 __half2char_dn

`char __half2char_dn(half a)`

This function converts type of `a` from `half` to `char` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

Cambricon@155chb

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__half2char](#) for more details.

3.11.135 __half2char_oz

`char __half2char_oz(half a)`

This function converts type of `a` from `half` to `char` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2char](#) for more details.

3.11.136 __half2char_rd

```
char __half2char_rd(half a)
```

This function converts type of a from `half` to `char` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2char](#) for more details.

3.11.137 __half2char_rm

```
char __half2char_rm(half a)
```

This function converts type of a from `half` to `char` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2char](#) for more details.

3.11.138 __half2char_rn

```
char __half2char_rn(half a)
```

This function converts type of a from `half` to `char` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2char](#) for more details.

3.11.139 __half2char_tz

```
char __half2char_tz(half a)
```

This function converts type of a from `half` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2char](#) for more details.

3.11.140 __half2char_up

```
char __half2char_up(half a)
```

This function converts type of a from `half` to `char` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2char](#) for more details.

3.11.141 __half2float

```
float __half2float(half a)
```

This function converts type of a from `half` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    float result;
    result = __half2float(a);
}
```

3.11.142 __half2float_dn

```
float __half2float_dn(half a)
```

This function converts type of a from `half` to `float` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2float](#) for more details.

3.11.143 __half2float_oz

```
float __half2float_oz(half a)
```

This function converts type of a from `half` to `float` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2float](#) for more details.

3.11.144 __half2float_rd

```
float __half2float_rd(half a)
```

This function converts type of `a` from `half` to `float` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2float](#) for more details.

3.11.145 __half2float_rm

```
float __half2float_rm(half a)
```

This function converts type of `a` from `half` to `float` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2float](#) for more details.

3.11.146 __half2float_rn

```
float __half2float_rn(half a)
```

This function converts type of a from `half` to `float` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2float](#) for more details.

3.11.147 __half2float_tz

```
float __half2float_tz(half a)
```

This function converts type of a from `half` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2float](#) for more details.

3.11.148 __half2float_up

```
float __half2float_up(half a)
```

This function converts type of a from `half` to `float` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2float](#) for more details.

3.11.149 __half2int

```
int __half2int(half a)
```

This function converts type of a from `half` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.150 __half2int_dn

```
int __half2int_dn(half a)
```

This function converts type of a from `half` to `int` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.151 __half2int_oz

```
int __half2int_oz(half a)
```

This function converts type of a from `half` to `int` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.152 __half2int_rd

```
int __half2int_rd(half a)
```

This function converts type of a from `half` to `int` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.153 __half2int_rm

```
int __half2int_rm(half a)
```

This function converts type of a from `half` to `int` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.154 __half2int_rn

```
int __half2int_rn(half a)
```

This function converts type of `a` from `half` to `int` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.155 __half2int_tz

```
int __half2int_tz(half a)
```

This function converts type of `a` from `half` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    int result;
    result = __half2int_tz(a);
}
```

3.11.156 __half2int_up

```
int __half2int_up(half a)
```

This function converts type of a from half to int in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2int_tz](#) for more details.

3.11.157 __half2short

```
short __half2short(half a)
```

This function converts type of a from `half` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__half2short_tz](#) for more details.

3.11.158 __half2short_dn

```
short __half2short_dn(half a)
```

This function converts type of a from `half` to `short` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__half2short_tz](#) for more details.

3.11.159 __half2short_oz

```
short __half2short_oz(half a)
```

This function converts type of a from `half` to `short` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__half2short_tz](#) for more details.

3.11.160 __half2short_rd

```
short __half2short_rd(half a)
```

This function converts type of a from `half` to `short` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__half2short_tz](#) for more details.

3.11.161 __half2short_rm

```
short __half2short_rm(half a)
```

This function converts type of a from `half` to `short` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__half2short_tz](#) for more details.

3.11.162 __half2short_rn

```
short __half2short_rn(half a)
```

This function converts type of a from `half` to `short` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__half2short_tz](#) for more details.

3.11.163 __half2short_tz

```
short __half2short_tz(half a)
```

This function converts type of a from `half` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>
__mlu_entry__ void kernel(half a) {
    short result;
    result = __half2short_tz(a);
}
```

3.11.164 __half2short_up

```
short __half2short_up(half a)
```

This function converts type of a from `half` to `short` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;

- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of `__half2short_tz` for more details.

3.11.165 `__half2tf32`

```
float __half2tf32(half a)
```

This function converts type of `a` from `half` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    float result;
    result = __half2tf32(a);
}
```

3.11.166 `__half2tf32_dn`

```
float __half2tf32_dn(half a)
```

This function converts type of `a` from `half` to `tf32` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.167 __half2tf32_oz

```
float __half2tf32_oz(half a)
```

This function converts type of a from half to tf32 in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.168 __half2tf32_rd

```
float __half2tf32_rd(half a)
```

This function converts type of a from half to tf32 in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.169 __half2tf32_rm

```
float __half2tf32_rm(half a)
```

This function converts type of a from half to tf32 in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.170 __half2tf32_rn

```
float __half2tf32_rn(half a)
```

This function converts type of a from half to tf32 in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.171 [__half2tf32_tz](#)

```
float __half2tf32_tz(half a)
```

This function converts type of a from `half` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.172 [__half2tf32_up](#)

```
float __half2tf32_up(half a)
```

This function converts type of a from `half` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__half2tf32](#) for more details.

3.11.173 [__half2uchar](#)

```
unsigned char __half2uchar(half a)
```

This function converts type of a from `half` to `unsigned char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    unsigned char result;
    result = __half2uchar(a);
}
```

3.11.174 __half2uchar_dn

```
unsigned char __half2uchar_dn(half a)
```

This function converts type of a from `half` to `unsigned char` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__half2uchar](#) for more details.

3.11.175 __half2uchar_oz

Cambricon@155chb

```
unsigned char __half2uchar_oz(half a)
```

This function converts type of a from `half` to `unsigned char` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__half2uchar](#) for more details.

3.11.176 __half2uchar_rd

```
unsigned char __half2uchar_rd(half a)
```

This function converts type of a from half to unsigned char in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2uchar](#) for more details.

3.11.177 __half2uchar_rm

```
unsigned char __half2uchar_rm(half a)
```

This function converts type of a from half to unsigned char in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2uchar](#) for more details.

3.11.178 __half2uchar_rn

```
unsigned char __half2uchar_rn(half a)
```

This function converts type of a from `half` to `unsigned char` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__half2uchar](#) for more details.

3.11.179 __half2uchar_tz

Cambricon@155chb

```
unsigned char __half2uchar_tz(half a)
```

This function converts type of a from `half` to `unsigned char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__half2uchar](#) for more details.

3.11.180 __half2uchar_up

```
unsigned char __half2uchar_up(half a)
```

This function converts type of a from `half` to `unsigned char` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__half2uchar](#) for more details.

3.11.181 __half2uint

```
unsigned int __half2uint(half a)
```

This function converts type of a from `half` to `unsigned int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    unsigned int result;
    result = __half2uint(a);
}
```

3.11.182 __half2uint_dn

`unsigned int __half2uint_dn(half a)`

This function converts type of `a` from `half` to `unsigned int` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

Cambricon@155chb

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__half2uint](#) for more details.

3.11.183 __half2uint_oz

`unsigned int __half2uint_oz(half a)`

This function converts type of `a` from `half` to `unsigned int` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2uint](#) for more details.

3.11.184 [__half2uint_rd](#)

```
unsigned int __half2uint_rd(half a)
```

This function converts type of a from `half` to `unsigned int` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2uint](#) for more details.

3.11.185 [__half2uint_rm](#)

```
unsigned int __half2uint_rm(half a)
```

This function converts type of a from `half` to `unsigned int` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2uint](#) for more details.

3.11.186 __half2uint_rn

```
unsigned int __half2uint_rn(half a)
```

This function converts type of a from `half` to `unsigned int` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2uint](#) for more details.

3.11.187 __half2uint_tz

```
unsigned int __half2uint_tz(half a)
```

This function converts type of a from `half` to `unsigned int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2uint](#) for more details.

3.11.188 __half2uint_up

```
unsigned int __half2uint_up(half a)
```

This function converts type of a from `half` to `unsigned int` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2uint](#) for more details.

3.11.189 __half2ushort

```
unsigned short __half2ushort(half a)
```

This function converts type of a from `half` to `unsigned short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(half a) {
    unsigned short result;
    result = __half2ushort(a);
}
```

3.11.190 __half2ushort_dn

unsigned short __half2ushort_dn(half a)

This function converts type of a from half to unsigned short in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

Cambricon@155chb

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.191 __half2ushort_oz

unsigned short __half2ushort_oz(half a)

This function converts type of a from half to unsigned short in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.192 __half2ushort_rd

```
unsigned short __half2ushort_rd(half a)
```

This function converts type of a from `half` to `unsigned short` in round-nearest-off-zero mode.

See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.193 __half2ushort_rm

```
unsigned short __half2ushort_rm(half a)
```

This function converts type of a from `half` to `unsigned short` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.194 __half2ushort_rn

```
unsigned short __half2ushort_rn(half a)
```

This function converts type of a from `half` to `unsigned short` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.195 __half2ushort_tz

```
unsigned short __half2ushort_tz(half a)
```

This function converts type of a from `half` to `unsigned short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.196 __half2ushort_up

```
unsigned short __half2ushort_up(half a)
```

This function converts type of a from `half` to `unsigned short` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__half2ushort](#) for more details.

3.11.197 __int2bfloat16

```
bfloat16_t __int2bfloat16(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(int a) {
    bfloat16_t result;
    result = __int2bfloat16(a);
}
```

3.11.198 __int2bfloat16_dn

bfloat16_t __int2bfloat16_dn(int a)

This function converts type of a from int to bfloat16_t in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.199 __int2bfloat16_oz

```
bfloat16_t __int2bfloat16_oz(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.200 __int2bfloat16_rd

```
bfloat16_t __int2bfloat16_rd(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.201 __int2bfloat16_rm

```
bfloat16_t __int2bfloat16_rm(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.202 __int2bfloat16_rn

Cambricon@155chb

```
bfloat16_t __int2bfloat16_rn(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.203 __int2bfloat16_tz

```
bfloat16_t __int2bfloat16_tz(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.204 __int2bfloat16_up

```
bfloat16_t __int2bfloat16_up(int a)
```

This function converts type of a from `int` to `bfloat16_t` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2bfloat16](#) for more details.

3.11.205 __int2float

```
float __int2float(int a)
```

This function converts type of a from `int` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>
__mlu_entry__ void kernel(int a) {
    float result;
    result = __int2float(a);
}
```

3.11.206 __int2float_dn

```
float __int2float_dn(int a)
```

This function converts type of a from `int` to `float` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.5.0`;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.207 __int2float_oz

```
float __int2float_oz(int a)
```

This function converts type of `a` from `int` to `float` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.208 __int2float_rd

```
float __int2float_rd(int a)
```

This function converts type of `a` from `int` to `float` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.209 __int2float_rm

```
float __int2float_rm(int a)
```

This function converts type of `a` from `int` to `float` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.210 __int2float_rn

```
float __int2float_rn(int a)
```

This function converts type of `a` from `int` to `float` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.5.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.211 __int2float_tz

```
float __int2float_tz(int a)
```

This function converts type of `a` from `int` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.212 __int2float_up

```
float __int2float_up(int a)
```

This function converts type of `a` from `int` to `float` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2float](#) for more details.

3.11.213 __int2half

```
half __int2half(int a)
```

This function converts type of `a` from `int` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2half_tz](#) for more details.

3.11.214 __int2half_dn

```
half __int2half_dn(int a)
```

This function converts type of `a` from `int` to `half` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2half_tz](#) for more details.

3.11.215 __int2half_oz

```
half __int2half_oz(int a)
```

This function converts type of `a` from `int` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2half_tz](#) for more details.

3.11.216 __int2half_rd

```
half __int2half_rd(int a)
```

This function converts type of `a` from `int` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__int2half_tz](#) for more details.

3.11.217 __int2half_rm

```
half __int2half_rm(int a)
```

This function converts type of `a` from `int` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__int2half_tz](#) for more details.

3.11.218 __int2half_rn

```
half __int2half_rn(int a)
```

This function converts type of `a` from `int` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.5.0;`

- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of `__int2half_tz` for more details.

3.11.219 __int2half_tz

```
half __int2half_tz(int a)
```

This function converts type of a from `int` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.5.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(int a) {
    half result;
    result = __int2half_tz(a);
}
```

3.11.220 __int2half_up

```
half __int2half_up(int a)
```

This function converts type of a from `int` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__int2half_tz](#) for more details.

3.11.221 __int2tf32

```
float __int2tf32(int a)
```

This function converts type of a from `int` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(int a) {
    float result;
    result = __int2tf32(a);
}
```

3.11.222 __int2tf32_dn

```
float __int2tf32_dn(int a)
```

This function converts type of `a` from `int` to `tf32` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.223 __int2tf32_oz

```
float __int2tf32_oz(int a)
```

This function converts type of `a` from `int` to `tf32` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.224 __int2tf32_rd

```
float __int2tf32_rd(int a)
```

This function converts type of a from `int` to `tf32` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.225 __int2tf32_rm

```
float __int2tf32_rm(int a)
```

This function converts type of a from `int` to `tf32` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.226 __int2tf32_rn

```
float __int2tf32_rn(int a)
```

This function converts type of a from `int` to `tf32` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.227 __int2tf32_tz

```
float __int2tf32_tz(int a)
```

This function converts type of a from `int` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.228 __int2tf32_up

```
float __int2tf32_up(int a)
```

This function converts type of `a` from `int` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__int2tf32](#) for more details.

3.11.229 __popcnt Cambricon@155chb

```
unsigned char __popcnt(unsigned char src)
```

```
unsigned short __popcnt(unsigned short src)
```

```
unsigned int __popcnt(unsigned int src)
```

Counts the number of bits that are set to 1 in `<src>`.

Parameters

- [in] `src`: The input data.

Return

- `unsigned char/short/int`.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.4.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.11.230 __short2bfloat16

```
bfloat16_t __short2bfloat16(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

Cambricon@155chb

```
#include <bang.h>

__mlu_entry__ void kernel(short a) {
    bfloat16_t result;
    result = __short2bfloat16(a);
}
```

3.11.231 __short2bfloat16_dn

```
bfloat16_t __short2bfloat16_dn(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.232 __short2bfloat16_oz

```
bfloat16_t __short2bfloat16_oz(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.233 __short2bfloat16_rd

```
bfloat16_t __short2bfloat16_rd(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.234 __short2bfloat16_rm

```
bfloat16_t __short2bfloat16_rm(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.235 __short2bfloat16_rn

```
bfloat16_t __short2bfloat16_rn(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.236 __short2bfloat16_tz

```
bfloat16_t __short2bfloat16_tz(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.237 __short2bfloat16_up

```
bfloat16_t __short2bfloat16_up(short a)
```

This function converts type of a from `short` to `bfloat16_t` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2bfloat16](#) for more details.

3.11.238 __short2float

```
float __short2float(short a)
```

This function converts type of a from `short` to `float`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.11.239 __short2half

```
half __short2half(short a)
```

This function converts type of a from `short` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__short2half_tz` for more details.

3.11.240 __short2half_dn

```
half __short2half_dn(short a)
```

This function converts type of `a` from `short` to `half` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__short2half_tz` for more details.

3.11.241 __short2half_oz

```
half __short2half_oz(short a)
```

This function converts type of `a` from `short` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__short2half_tz` for more details.

3.11.242 __short2half_rd

```
half __short2half_rd(short a)
```

This function converts type of `a` from `short` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__short2half_tz` for more details.

3.11.243 __short2half_rm

```
half __short2half_rm(short a)
```

This function converts type of `a` from `short` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__short2half_tz` for more details.

3.11.244 `__short2half_rn`

```
half __short2half_rn(short a)
```

This function converts type of `a` from `short` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of `__short2half_tz` for more details.

3.11.245 `__short2half_tz`

```
half __short2half_tz(short a)
```

This function converts type of `a` from `short` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(short a) {
    half result;
    result = __short2half_tz(a);
}
```

3.11.246 __short2half_up

`half __short2half_up(short a)`

This function converts type of `a` from `short` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of `__short2half_tz` for more details.

3.11.247 __short2tf32

`float __short2tf32(short a)`

This function converts type of `a` from `short` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(short a) {
    float result;
    result = __short2tf32(a);
}
```

3.11.248 __short2tf32_dn

```
float __short2tf32_dn(short a)
```

This function converts type of a from short to tf32 in round-down mode. See the table Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__short2tf32](#) for more details.

3.11.249 __short2tf32_oz

```
float __short2tf32_oz(short a)
```

This function converts type of a from `short` to `tf32` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__short2tf32](#) for more details.

3.11.250 __short2tf32_rd

```
float __short2tf32_rd(short a)
```

This function converts type of a from `short` to `tf32` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__short2tf32](#) for more details.

3.11.251 __short2tf32_rm

```
float __short2tf32_rm(short a)
```

This function converts type of a from `short` to `tf32` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__short2tf32](#) for more details.

3.11.252 __short2tf32_rn

```
float __short2tf32_rn(short a)
```

This function converts type of a from `short` to `tf32` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__short2tf32](#) for more details.

3.11.253 __short2tf32_tz

```
float __short2tf32_tz(short a)
```

This function converts type of a from `short` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__short2tf32](#) for more details.

3.11.254 __short2tf32_up

```
float __short2tf32_up(short a)
```

This function converts type of a from `short` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__short2tf32](#) for more details.

3.11.255 __tf322bf16

```
bf16_t __tf322bf16(float a)
```

This function converts type of a from tf32 to bf16_t in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>
__mlu_entry__ void kernel(float a) {
    bf16_t result;
    result = __tf322bf16(a);
}
```

3.11.256 __tf322bf16_dn

```
bf16_t __tf322bf16_dn(float a)
```

This function converts type of a from tf32 to bf16_t in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of `__tf322bfloat16` for more details.

3.11.257 __tf322bfloat16_oz

```
bfloat16_t __tf322bfloat16_oz(float a)
```

This function converts type of `a` from `tf32` to `bfloat16_t` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of `__tf322bfloat16` for more details.

3.11.258 __tf322bfloat16_rd

```
bfloat16_t __tf322bfloat16_rd(float a)
```

This function converts type of `a` from `tf32` to `bfloat16_t` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__tf322bfloat16](#) for more details.

3.11.259 __tf322bfloat16_rm

```
bfloat16_t __tf322bfloat16_rm(float a)
```

This function converts type of `a` from `tf32` to `bfloat16_t` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__tf322bfloat16](#) for more details.

3.11.260 __tf322bfloat16_rn

```
bfloat16_t __tf322bfloat16_rn(float a)
```

This function converts type of `a` from `tf32` to `bfloat16_t` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of `__tf322bfloat16` for more details.

3.11.261 __tf322bfloat16_tz

```
bfloat16_t __tf322bfloat16_tz(float a)
```

This function converts type of `a` from `tf32` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of `__tf322bfloat16` for more details.

3.11.262 __tf322bfloat16_up

```
bfloat16_t __tf322bfloat16_up(float a)
```

This function converts type of `a` from `tf32` to `bfloat16_t` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`

- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322bfloat16](#) for more details.

3.11.263 __tf322char

```
char __tf322char(float a)
```

This function converts type of `a` from `tf32` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    char result;
    result = __tf322char(a);
}
```

3.11.264 __tf322char_dn

```
char __tf322char_dn(float a)
```

This function converts type of `a` from `tf32` to `char` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.265 __tf322char_oz

```
char __tf322char_oz(float a)
```

This function converts type of a from `tf32` to `char` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.266 __tf322char_rd

```
char __tf322char_rd(float a)
```

This function converts type of a from `tf32` to `char` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.267 __tf322char_rm

```
char __tf322char_rm(float a)
```

This function converts type of a from `tf32` to `char` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.268 __tf322char_rn

```
char __tf322char_rn(float a)
```

This function converts type of a from `tf32` to `char` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.269 __tf322char_tz

```
char __tf322char_tz(float a)
```

This function converts type of a from `tf32` to `char` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.270 __tf322char_up

```
char __tf322char_up(float a)
```

This function converts type of a from `tf32` to `char` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322char](#) for more details.

3.11.271 __tf322float

```
float __tf322float(float a)
```

This function converts type of a from `tf32` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Cambricon@155chb

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    float result;
    result = __tf322float(a);
}
```

3.11.272 __tf322float_dn

```
float __tf322float_dn(float a)
```

This function converts type of a from `tf32` to `float` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf322float](#) for more details.

3.11.273 __tf322float_oz

```
float __tf322float_oz(float a)
```

This function converts type of a from `tf32` to `float` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf322float](#) for more details.

3.11.274 __tf322float_rd

```
float __tf322float_rd(float a)
```

This function converts type of a from `tf32` to `float` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf322float](#) for more details.

3.11.275 __tf322float_rm

```
float __tf322float_rm(float a)
```

This function converts type of a from `tf32` to `float` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf322float](#) for more details.

3.11.276 __tf322float_rn

```
float __tf322float_rn(float a)
```

This function converts type of a from `tf32` to `float` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf322float](#) for more details.

3.11.277 __tf322float_tz

```
float __tf322float_tz(float a)
```

This function converts type of a from `tf32` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf322float](#) for more details.

3.11.278 __tf32float_up

```
float __tf32float_up(float a)
```

This function converts type of a from `tf32` to `float` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__tf32float](#) for more details.

3.11.279 __tf32half

Cambricon@155chb

```
half __tf32half(float a)
```

This function converts type of a from `tf32` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    half result;
    result = __tf32half(a);
}
```

3.11.280 __tf32half_dn

half __tf32half_dn(float a)

This function converts type of a from tf32 to half in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

Cambricon@155chb

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32half](#) for more details.

3.11.281 __tf32half_oz

half __tf32half_oz(float a)

This function converts type of a from tf32 to half in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32half](#) for more details.

3.11.282 __tf32half_rd

```
half __tf32half_rd(float a)
```

This function converts type of a from `tf32` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32half](#) for more details.

3.11.283 __tf32half_rm

```
half __tf32half_rm(float a)
```

This function converts type of a from `tf32` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322half](#) for more details.

3.11.284 __tf322half_rn

```
half __tf322half_rn(float a)
```

This function converts type of a from `tf32` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322half](#) for more details.

3.11.285 __tf322half_tz

```
half __tf322half_tz(float a)
```

This function converts type of a from `tf32` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322half](#) for more details.

3.11.286 __tf322half_up

```
half __tf322half_up(float a)
```

This function converts type of a from `tf32` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322half](#) for more details.

3.11.287 __tf322int

```
int __tf322int(float a)
```

This function converts type of a from `tf32` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    int result;
    result = __tf32int(a);
}
```

3.11.288 __tf32int_dn

```
int __tf32int_dn(float a)
```

This function converts type of a from tf32 to int in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.289 __tf32int_oz

```
int __tf32int_oz(float a)
```

This function converts type of a from tf32 to int in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.290 __tf32int_rd

```
int __tf32int_rd(float a)
```

This function converts type of a from `tf32` to `int` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.291 __tf32int_rm

```
int __tf32int_rm(float a)
```

This function converts type of a from `tf32` to `int` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.292 __tf32int_rn

```
int __tf32int_rn(float a)
```

This function converts type of `a` from `tf32` to `int` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.293 __tf32int_tz

```
int __tf32int_tz(float a)
```

This function converts type of `a` from `tf32` to `int` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.294 __tf32int_up

```
int __tf32int_up(float a)
```

This function converts type of a from `tf32` to `int` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf32int](#) for more details.

3.11.295 __tf32short

```
short __tf32short(float a)
```

This function converts type of a from `tf32` to `short` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(float a) {
    short result;
    result = __tf322short(a);
}
```

3.11.296 __tf322short_dn

```
short __tf322short_dn(float a)
```

This function converts type of a from tf32 to short in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322short](#) for more details.

3.11.297 __tf322short_oz

```
short __tf322short_oz(float a)
```

This function converts type of a from `tf32` to `short` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__tf322short](#) for more details.

3.11.298 __tf322short_rd

```
short __tf322short_rd(float a)
```

This function converts type of a from `tf32` to `short` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__tf322short](#) for more details.

3.11.299 __tf322short_rm

```
short __tf322short_rm(float a)
```

This function converts type of a from tf32 to short in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322short](#) for more details.

3.11.300 __tf322short_rn

```
short __tf322short_rn(float a)
```

This function converts type of a from tf32 to short in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322short](#) for more details.

3.11.301 __tf322short_tz

```
short __tf322short_tz(float a)
```

This function converts type of a from tf32 to short in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322short](#) for more details.

3.11.302 __tf322short_up

```
short __tf322short_up(float a)
```

This function converts type of a from tf32 to short in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__tf322short](#) for more details.

3.11.303 __uchar2bfloat16

```
bfloat16_t __uchar2bfloat16(unsigned char a)
```

This function converts type of a from `unsigned char` to `bfloat16_t`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>
__mlu_entry__ void kernel(unsigned char a) {
    bfloat16_t result;
    result = __uchar2bfloat16(a);
}
```

3.11.304 __uchar2float

```
float __uchar2float(unsigned char a)
```

This function converts type of a from `unsigned char` to `float`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.11.305 __uchar2half

```
half __uchar2half(unsigned char a)
```

This function converts type of a from `unsigned char` to `half`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.11.306 __uchar2tf32

```
float __uchar2tf32(unsigned char a)
```

This function converts type of a from `unsigned char` to `tf32`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(unsigned char a) {
    float result;
    result = __uchar2tf32(a);
}
```

3.11.307 __uint2bfloat16

`bfloat16_t __uint2bfloat16(unsigned int a)`

This function converts type of `a` from `unsigned int` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(unsigned int a) {
    bfloat16_t result;
    result = __uint2bfloat16(a);
}
```

3.11.308 __uint2bfloat16_dn

```
bfloat16_t __uint2bfloat16_dn(unsigned int a)
```

This function converts type of `a` from `unsigned int` to `bfloat16_t` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.309 __uint2bfloat16_oz

Cambricon@155chb

```
bfloat16_t __uint2bfloat16_oz(unsigned int a)
```

This function converts type of `a` from `unsigned int` to `bfloat16_t` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.310 __uint2bfloat16_rd

```
bfloat16_t __uint2bfloat16_rd(unsigned int a)
```

This function converts type of a from `unsigned int` to `bfloat16_t` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.311 __uint2bfloat16_rm

```
bfloat16_t __uint2bfloat16_rm(unsigned int a)
```

This function converts type of a from `unsigned int` to `bfloat16_t` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.312 __uint2bfloat16_rn

```
bfloat16_t __uint2bfloat16_rn(unsigned int a)
```

This function converts type of a from `unsigned int` to `bfloat16_t` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.313 __uint2bfloat16_tz

Cambricon@155chb

```
bfloat16_t __uint2bfloat16_tz(unsigned int a)
```

This function converts type of a from `unsigned int` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.314 __uint2bfloat16_up

```
bfloat16_t __uint2bfloat16_up(unsigned int a)
```

This function converts type of a from `unsigned int` to `bfloat16_t` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2bfloat16](#) for more details.

3.11.315 __uint2float

Cambricon@155chb

```
float __uint2float(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__uint2float_tz](#) for more details.

3.11.316 __uint2float_dn

```
float __uint2float_dn(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__uint2float_tz](#) for more details.

3.11.317 __uint2float_oz

Cambricon@155chb

```
float __uint2float_oz(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__uint2float_tz](#) for more details.

3.11.318 __uint2float_rd

```
float __uint2float_rd(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__uint2float_tz](#) for more details.

3.11.319 __uint2float_rm

Cambricon@155chb

```
float __uint2float_rm(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__uint2float_tz](#) for more details.

3.11.320 __uint2float_rn

```
float __uint2float_rn(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- See the example of [__uint2float_tz](#) for more details.

3.11.321 __uint2float_tz

Cambricon@155chb

```
float __uint2float_tz(unsigned int a)
```

This function converts type of a from `unsigned int` to `float` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(unsigned int a) {
    float result;
    result = __uint2float_tz(a);
}
```

3.11.322 __uint2float_up

`float __uint2float_up(unsigned int a)`

This function converts type of `a` from `unsigned int` to `float` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

Cambricon@155chb

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.5.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__uint2float_tz` for more details.

3.11.323 __uint2half

`half __uint2half(unsigned int a)`

This function converts type of `a` from `unsigned int` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__uint2half_tz](#) for more details.

3.11.324 __uint2half_dn

```
half __uint2half_dn(unsigned int a)
```

This function converts type of a from `unsigned int` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__uint2half_tz](#) for more details.

3.11.325 __uint2half_oz

```
half __uint2half_oz(unsigned int a)
```

This function converts type of a from `unsigned int` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__uint2half_tz](#) for more details.

3.11.326 __uint2half_rd

```
half __uint2half_rd(unsigned int a)
```

This function converts type of a from `unsigned int` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__uint2half_tz](#) for more details.

3.11.327 __uint2half_rm

```
half __uint2half_rm(unsigned int a)
```

This function converts type of a from `unsigned int` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__uint2half_tz](#) for more details.

3.11.328 __uint2half_rn

```
half __uint2half_rn(unsigned int a)
```

This function converts type of a from `unsigned int` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- See the example of [__uint2half_tz](#) for more details.

3.11.329 __uint2half_tz

```
half __uint2half_tz(unsigned int a)
```

This function converts type of a from `unsigned int` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(int a) {
    half result;
    result = __uint2half_tz(a);
}
```

3.11.330 __uint2half_up

`half __uint2half_up(unsigned int a)`

This function converts type of a from `unsigned int` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of `__uint2half_tz` for more details.

3.11.331 __uint2tf32

`float __uint2tf32(unsigned int a)`

This function converts type of a from `unsigned int` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(unsigned int a) {
    float result;
    result = __uint2tf32(a);
}
```

3.11.332 __uint2tf32_dn

float __uint2tf32_dn(unsigned int a)

This function converts type of a from unsigned int to tf32 in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__uint2tf32](#) for more details.

3.11.333 __uint2tf32_oz

```
float __uint2tf32_oz(unsigned int a)
```

This function converts type of a from `unsigned int` to `tf32` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2tf32](#) for more details.

3.11.334 __uint2tf32_rd

Cambricon@155chb

```
float __uint2tf32_rd(unsigned int a)
```

This function converts type of a from `unsigned int` to `tf32` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2tf32](#) for more details.

3.11.335 __uint2tf32_rm

```
float __uint2tf32_rm(unsigned int a)
```

This function converts type of a from `unsigned int` to `tf32` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2tf32](#) for more details.

3.11.336 __uint2tf32_rn

Cambricon@155chb

```
float __uint2tf32_rn(unsigned int a)
```

This function converts type of a from `unsigned int` to `tf32` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2tf32](#) for more details.

3.11.337 __uint2tf32_tz

```
float __uint2tf32_tz(unsigned int a)
```

This function converts type of a from `unsigned int` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2tf32](#) for more details.

3.11.338 __uint2tf32_up

Cambricon@155chb

```
float __uint2tf32_up(unsigned int a)
```

This function converts type of a from `unsigned int` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__uint2tf32](#) for more details.

3.11.339 __ushort2bfloat16

```
bfloat16_t __ushort2bfloat16(unsigned short a)
```

This function converts type of a from `unsigned short` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>
__mlu_entry__ void kernel(unsigned short a) {
    bfloat16_t result;
    result = __ushort2bfloat16(a);
}
```

3.11.340 __ushort2bfloat16_dn

```
bfloat16_t __ushort2bfloat16_dn(unsigned short a)
```

This function converts type of a from `unsigned short` to `bfloat16_t` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.341 __ushort2bfloat16_oz

```
bfloat16_t __ushort2bfloat16_oz(unsigned short a)
```

This function converts type of a from `unsigned short` to `bfloat16_t` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.342 __ushort2bfloat16_rd

```
bfloat16_t __ushort2bfloat16_rd(unsigned short a)
```

This function converts type of a from `unsigned short` to `bfloat16_t` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.343 __ushort2bfloat16_rm

```
bfloat16_t __ushort2bfloat16_rm(unsigned short a)
```

This function converts type of `a` from `unsigned short` to `bfloat16_t` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.344 __ushort2bfloat16_rn

```
bfloat16_t __ushort2bfloat16_rn(unsigned short a)
```

This function converts type of `a` from `unsigned short` to `bfloat16_t` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] `a`: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.345 __ushort2bfloat16_tz

```
bfloat16_t __ushort2bfloat16_tz(unsigned short a)
```

This function converts type of a from `unsigned short` to `bfloat16_t` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.346 __ushort2bfloat16_up

```
bfloat16_t __ushort2bfloat16_up(unsigned short a)
```

This function converts type of a from `unsigned short` to `bfloat16_t` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2bfloat16](#) for more details.

3.11.347 __ushort2float

```
float __ushort2float(unsigned short a)
```

This function converts type of a from `unsigned short` to `float`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.11.348 __ushort2half

```
half __ushort2half(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__ushort2half_tz](#) for more details.

3.11.349 __ushort2half_dn

```
half __ushort2half_dn(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0,`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__ushort2half_tz](#) for more details.

3.11.350 __ushort2half_oz

```
half __ushort2half_oz(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__ushort2half_tz](#) for more details.

3.11.351 __ushort2half_rd

```
half __ushort2half_rd(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__ushort2half_tz](#) for more details.

3.11.352 __ushort2half_rm

```
half __ushort2half_rm(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__ushort2half_tz](#) for more details.

3.11.353 __ushort2half_rn

```
half __ushort2half_rn(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0,`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__ushort2half_tz](#) for more details.

3.11.354 __ushort2half_tz

```
half __ushort2half_tz(unsigned short a)
```

This function converts type of a from `unsigned short` to `half` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`

- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

__mlu_entry__ void kernel(unsigned short a) {
    half result;
    result = __ushort2half_tz(a);
}
```

3.11.355 __ushort2half_up

`half __ushort2half_up(unsigned short a)`

This function converts type of a from `unsigned short` to `half` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 3.5.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__ushort2half_tz` for more details.

3.11.356 __ushort2tf32

`float __ushort2tf32(unsigned short a)`

This function converts type of a from `unsigned short` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

__mlu_entry__ void kernel(unsigned short a) {
    float result;
    result = __ushort2tf32(a);
}
```

3.11.357 __ushort2tf32_dn

```
float __ushort2tf32_dn(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-down mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- See the example of [__ushort2tf32](#) for more details.

3.11.358 __ushort2tf32_oz

```
float __ushort2tf32_oz(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2tf32](#) for more details.

3.11.359 __ushort2tf32_rd

Cambricon@155chb

```
float __ushort2tf32_rd(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-nearest-off-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2tf32](#) for more details.

3.11.360 __ushort2tf32_rm

```
float __ushort2tf32_rm(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-math mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2tf32](#) for more details.

3.11.361 __ushort2tf32_rm

Cambricon@155chb

```
float __ushort2tf32_rm(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-nearest-even mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2tf32](#) for more details.

3.11.362 __ushort2tf32_tz

```
float __ushort2tf32_tz(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-to-zero mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2tf32](#) for more details.

3.11.363 __ushort2tf32_up

Cambricon@155chb

```
float __ushort2tf32_up(unsigned short a)
```

This function converts type of a from `unsigned short` to `tf32` in round-up mode. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [in] a: The source data.

Return

- The result of conversion.

Remark

- None.

Instruction Pipeline

- Scalar.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- See the example of [__ushort2tf32](#) for more details.

3.12 Synchronization Functions

3.12.1 __sync_all

```
void __sync_all()
```

Synchronizes all clusters on which the kernel executes. Its effect is equal to __sync_all_ipu + __sync_all_mpu. The number of MLU core and mpu core to synchronize is related to task type, UNION1 is 5, UNION2 is 10, UNION4 is 20.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.2 __sync_all_ipu

```
void __sync_all_ipu()
```

Synchronizes all MLU cores of clusters on which the kernel executes. The number of MLU cores to synchronize is related to task type, e.g., UNION1 is 4, UNION2 is 8, and UNION4 is 16.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.3 __sync_all_mpu

```
void __sync_all_mpu()
```

Synchronizes all MPU cores of clusters on which the kernel executes. The number of MPU cores to synchronize is related to task type, e.g., UNION1 is 1, UNION2 is 2, and UNION4 is 4.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.4 __sync_cluster

```
void __sync_cluster()
```

Synchronizes all cores inside the cluster on which the kernel executes. The cores to synchronize consists of 1 mpu core and 4 MLU cores.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.5 __sync_compute

```
void __sync_compute()
```

Synchronizes all instructions in compute pipeline within a core. All IO/Move/Compute instructions after this function must wait until all instructions in Compute pipeline have finished.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.6 __sync_io

```
void __sync_io()
```

Cambricon@155chb

Synchronizes all instructions in IO pipeline within a core. All IO/Move/Compute instructions after this function must wait until all instructions in IO pipeline have finished.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.7 __sync_io_move_compute

```
void __sync_io_move_compute()
```

Synchronizes all instructions in IO/Move/Compute pipelines within a core. All IO/Move/Compute instructions after this function must wait until all instructions in IO/Move/Compute pipelines have finished.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.12.8 __sync_move

```
void __sync_move()
```

Cambricon@155chb

Synchronizes all instructions in move pipeline within a MLU core. All IO/Move/Compute instructions after this function must wait until all instructions in Move pipeline have finished.

Return void.

Remark

- None.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13 Vector Active Functions

3.13.1 __bang_active_abs

```
void __bang_active_abs(half *dst,
                      half *src,
                      int elem_count)
```

```
void __bang_active_abs(float *dst,
                      float *src,
                      int elem_count)
```

This function computes the absolute value of each element in vector <src> and saves the result to vector <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: 0;
- <src> and <dst> must point to __nram__ space;
- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src>;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The range of <src> operand is [-65504, 65504] when type is half, and [-3.4 * 10³⁸, 3.4 * 10³⁸] when type is float.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.2 __bang_active_cos

```
void __bang_active_cos(half *dst,
                      half *src,
                      int elem_count)
```

```
void __bang_active_cos(float *dst,
                      float *src,
                      int elem_count)
```

Applies active (cosine) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: within 0.01;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The element value of <src> is in the range [- 2π , 2π] radian;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.3 __bang_active_exp

```
void __bang_active_exp(half *dst,
                      half *src,
                      int elem_count)
```

```
void __bang_active_exp(float *dst,
                      float *src,
                      int elem_count)
```

Applies active (exponent) operation on source operand <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: within 0.01;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The result is 5.96e-8, when the element value of <src> is less than -7.75;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the element value is in the range [-3.4e38, 10.25]; when the data type of the elements of <src> is half, the element value is in the range [-65504, 10.25].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.4 __bang_active_exp_less_0

```
void __bang_active_exp_less_0(half *dst,
                               half *src,
                               int elem_count)
```

```
void __bang_active_exp_less_0(float *dst,
                               float *src,
                               int elem_count)
```

Applies active (exponent) operation on source operand <src> when the value of <src> is less than 0.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The result is 5.96e-8, when the element value of <src> is less than -15.5;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the element value is in the range [-3.4e38, 0]; when the data type of the elements of <src> is half, the element value is in the range [-65504, 0];
- When the data type of the elements of <src> is float, the average relative error is within 0.003; when the data type of the elements of <src> is half, the average relative error is within 0.01;

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.5 __bang_active_exph

```
void __bang_active_exph(half *dst,
                        half *src,
                        int elem_count)

void __bang_active_exph(float *dst,
                        float *src,
                        int elem_count)
```

Applies high precision active (exponent) operation on source operand <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the element value is in the range

[-3.4e38, 16]; when the data type of the elements of <src> is `half`, the element value is in the range [-65504, 10.25];

- When the element value of <src> is less than -15 for the data type of `float`, the result is 3e-7; when the element value of <src> is less than -7.75 for the data type of `half`, the result is 5.96e-8;
- When the data type of the elements of <src> is `float`, the average relative error is within 0.003; when the data type of the elements of <src> is `half`, the average relative error is within 0.007.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.13.6 __bang_active_gelu

```
void __bang_active_gelu(half *dst,
                        half *src,
                        int elem_count)
void __bang_active_gelu(float *dst,
                        float *src,
                        int elem_count)
```

Applies active (gelu) operation on <src>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source vector.

Return

- `void`.

Remark

- Average relative error: within 0.003;
- <code>elem_count</code> must be greater than zero;
- <code>dst</code> can be overlapped with <code>src</code>;
- <code>src</code> and <code>dst</code> must point to `_nram_` address space;
- <code>elem_count * sizeof(type)</code> must be 128-byte aligned on `(m)tp_2xx`;
- The address of <code>src</code> and <code>dst</code> must be 64-byte aligned on `(m)tp_2xx`;
- When the data type of the elements of <code>src</code> is `float`, the element value is in the range [-3.4e38, 3.4e38]; when the data type of the elements of <code>src</code> is `half`, the element value is in the range [-65504, 65504];
- When the element value of <code>src</code> is less than -3.875, the result is -5.96e-8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.7 __bang_active_gelup

```
void __bang_active_gelup(half *dst,
                         half *src,
                         int elem_count)

void __bang_active_gelup(float *dst,
                         float *src,
                         int elem_count)
```

Applies active (gelup) operation on <src>, which has higher precision than active(gelu) does.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The result is -5.96e-8, when the element value of <src> is less than -3.875;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the element values of <src> are in the range [-3.4e38, 3.4e38]; when the data type of the elements of <src> is half, the element value is in the range [-65504, 65504];
- When the data type of the elements of <src> is float, the average relative error rate is within 0.003; when the data type of the elements of <src> is half, the average relative error rate is within 0.00411. Despite having higher average relative error than normal gelu operation, it has higher precision on critical interval.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.13.8 __bang_active_log

```
void __bang_active_log(half *dst,
                      half *src,
                      int elem_count)

void __bang_active_log(float *dst,
                      float *src,
                      int elem_count)
```

Applies active (log base e) operation on <src>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source vector.

Return

- `void`.

Remark

- Average relative error: within 0.01;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to `_nram_` address space;
- <dst> can be overlapped with <src>;
- The element value of <src> is in the range [6.1e-5, 63484];
- <elem_count> * `sizeof(type)` must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.13.9 __bang_active_loghp

```
void __bang_active_loghp(half *dst,
                         half *src,
                         int elem_count)
```

```
void __bang_active_loghp(float *dst,
                         float *src,
                         int elem_count)
```

Applies high precision active (log base e) operation on source operand <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the element value is in the range [7.2e-9, 507903]; when the data type of the elements of <src> is half, the element value is in the range [6.1e-5, 65504];
- When the data type of the elements of <src> is float, the average relative error is within 0.001; when the data type of the elements of <src> is half, the average relative error is within 0.002.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.10 __bang_active_pow2

```
void __bang_active_pow2(half *dst,
                        half *src,
                        int elem_count)

void __bang_active_pow2(float *dst,
                        float *src,
                        int elem_count)
```

Applies active (pow2) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: 0;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the element of <src> is half , the element values are integers in the range [-24.0, 15.0] , such as -24.0, -23.0, ... 0.0, 1.0, 2.0, ... 15.0 ; when the data type of the element of <src> is float, the element values are integers in the range [-31.0, 31.0], such as -31.0, -30.0, ... 0.0, 1.0, 2.0, ... 31.0.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.11 __bang_active_recip

```
void __bang_active_recip(half *dst,
                         half *src,
                         int elem_count)
```

```
void __bang_active_recip(float *dst,
                         float *src,
                         int elem_count)
```

Applies active (reciprocal) operation on <src>.

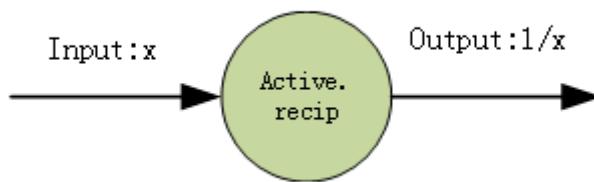


Fig. 3.5: Reciprocal Process

Hint:

Recip means reciprocal.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src>;
- <src> and <dst> must point to __nram__ address space;
- The element values of <src> are in the range [0.0078125, 65472];
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the average relative error is within 0.003; when the data type of the elements of <src> is half, the average relative error is within 0.01.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;

- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define LEN 64

__mlu_entry__ void kernel(half* y, half* x) {
    __nram__ half ny[LEN];
    __nram__ half nx[LEN];
    __memcpy(ny, y, LEN * sizeof(half), GDRAM2NRAM);
    __memcpy(nx, x, LEN * sizeof(half), GDRAM2NRAM);
    __bang_active_recip(ny, nx, LEN);
    __memcpy(y, ny, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.13.12 __bang_active_recip_greater_1

```
void __bang_active_recip_greater_1(half *dst,
                                    half *src,
                                    int elem_count)
```

```
void __bang_active_recip_greater_1(float *dst,
                                    float *src,
                                    int elem_count)
```

Applies active (reciprocal) operation on <src> when the value of <src> is greater than 1.

Hint:

Recip means reciprocal.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The element values of <src> are in the range [1, 63487];
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the average relative error is within 0.003; when the data type of the elements of <src> is half, the average relative error is within 0.01.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.13 __bang_active_reciphp

```
void __bang_active_reciphp(half *dst,
                            half *src,
                            int elem_count)
```

```
void __bang_active_reciphp(float *dst,
                           float *src,
                           int elem_count)
```

Applies high precision active (reciprocal) operation on source operand <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- Average relative error: within 0.0004;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- When the data type of the elements of <src> is float, the element value is in the range [2.2205e-16, 2e6]; when the data type of the elements of <src> is half, the element value is in the range [0.00391, 65504].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.14 __bang_active_relu

```
void __bang_active_relu(half *dst,
                        half *src,
                        int elem_count)

void __bang_active_relu(float *dst,
                        float *src,
                        int elem_count)
```

Applies active (relu) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: 0;
- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src>;
- <src> and <dst> must point to __nram__ address space;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The range of <src> operand is [-65504, 65504] when type is half, and [-3.4 * 10³⁸, 3.4 * 10³⁸] when type is float.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.15 __bang_active_rsqrt

```
void __bang_active_rsqrt(half *dst,
                        half *src,
                        int elem_count)

void __bang_active_rsqrt(float *dst,
                        float *src,
                        int elem_count)
```

Applies active (rsqrt) operation on <src>, $<dst> = 1 \div \sqrt{<src>}.$

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: within 0.01;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The element value of <src> is in the range [0.005, 63487];
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

Cambricon@155chb

3.13.16 __bang_active_rsqrthp

```
void __bang_active_rsqrthp(half *dst,
                            half *src,
                            int elem_count)

void __bang_active_rsqrthp(float *dst,
                           float *src,
                           int elem_count)
```

Applies high precision active (rsqrt) operation on source operand <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: within 0.0005;
- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src>;
- <src> and <dst> must point to __nram__ address space;

- `<elem_count> * sizeof(type)` must be 128-byte aligned on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- When the data type of the elements of `<src>` is `float`, the element value is in the range `[0.000367, 1e6]`; when the data type of the elements of `<src>` is `half`, the element value is in the range `[0.000367, 65504]`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.13.17 __bang_active_sigmoid

```
void __bang_active_sigmoid(half *dst,
                           half *src,
                           int elem_count)
```

```
void __bang_active_sigmoid(float *dst,
                           float *src,
                           int elem_count)
```

Applies active (sigmoid) operation on `<src>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source vector.

Return

- `void`.

Remark

- Average relative error: within 0.01;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `_nram_` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count> * sizeof(type)` must be 128-byte aligned on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- The range of `<src>` operand is `[-65504, 65504]` when type is `half`, and `[-3.4 * 1038, 3.4 * 1038]` when type is `float`;
- The result is 0, when the element value of `<src>` is less than -7.75; the result is 1, when the element value of `<src>` is greater than 7.75.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.13.18 __bang_active_sign

```
void __bang_active_sign(half *dst,
                        half *src,
                        int elem_count)

void __bang_active_sign(float *dst,
                        float *src,
                        int elem_count)
```

Applies active (sign) operation on `<src>`. If $<src> \geq 0.0$, then, stores 1.0 in `<dst>`; otherwise, stores -1.0 in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source vector.

Return

- `void`.

Remark

- Average relative error: 0;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count> * sizeof(type)` must be 128-byte aligned on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- When the element value of `<src>` is -0 or +0, the corresponding result is 1;
- When the data type of the elements of `<src>` is `float`, the element value is in the range [-3.4e38, 3.4e38]; when the data type of the elements of `<src>` is `half`, the element value is in the range [-65504, 65504].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.13.19 __bang_active_sin

```
void __bang_active_sin(half *dst,
                      half *src,
                      int elem_count)
```

```
void __bang_active_sin(float *dst,
                      float *src,
                      int elem_count)
```

Applies active (sine) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The element value of <src> is in the range [- 2π , 2π] radian;
- Average relative error: within 0.01;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.20 __bang_active_sqrt

```
void __bang_active_sqrt(half *dst,
                       half *src,
                       int elem_count)
```

```
void __bang_active_sqrt(float *dst,
                       float *src,
                       int elem_count)
```

Applies active (sqrt) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: within 0.01;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The element value of <src> is in the range [0, 63487];
- <elem_count> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

Cambricon@155chb

3.13.21 __bang_active_sqrthp

```
void __bang_active_sqrthp(half *dst,
                           half *src,
                           int elem_count)

void __bang_active_sqrthp(float *dst,
                           float *src,
                           int elem_count)
```

Applies high precision active (exponent) operation on source operand <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- Average relative error: within 0.003;
- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;

- $\langle \text{elem_count} \rangle * \text{sizeof}(\text{type})$ must be 128-byte aligned on $(\text{m})\text{tp_2xx}$;
- The address of $\langle \text{src} \rangle$ and $\langle \text{dst} \rangle$ must be 64-byte aligned on $(\text{m})\text{tp_2xx}$;
- When the data type of the elements of $\langle \text{src} \rangle$ is `float`, the element value is in the range $[0, 677268]$; when the data type of the elements of $\langle \text{src} \rangle$ is `half`, the element value is in the range $[0, 65504]$.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (\text{m})\text{tp_2xx}`.

Example

- None.

3.13.22 __bang_active_tanh

```
void __bang_active_tanh(half *dst,
                        half *src,
                        int elem_count)
```

```
void __bang_active_tanh(float *dst,
                        float *src,
                        int elem_count)
```

Applies active (`tanh`) operation on $\langle \text{src} \rangle$.

Parameters

- [out] dst : The address of destination vector.
- [in] src : The address of source vector.
- [in] elem_count : Number of elements in source vector.

Return

- `void`.

Remark

- Average relative error: within 0.01;
- $\langle \text{elem_count} \rangle$ must be greater than zero;
- $\langle \text{src} \rangle$ and $\langle \text{dst} \rangle$ must point to `_nram_` address space;
- $\langle \text{dst} \rangle$ can be overlapped with $\langle \text{src} \rangle$;
- $\langle \text{elem_count} \rangle * \text{sizeof}(\text{type})$ must be 128-byte aligned on $(\text{m})\text{tp_2xx}$;
- The address of $\langle \text{src} \rangle$ and $\langle \text{dst} \rangle$ must be 64-byte aligned on $(\text{m})\text{tp_2xx}$;
- When the data type of the elements of $\langle \text{src} \rangle$ is `float`, the element value is in the range $[-3.4e38, 3.4e38]$; when the data type of the elements of $\langle \text{src} \rangle$ is `half`, the element value is in the range $[-65504, 65504]$.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;

- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.13.23 __bang_taylor3_cos

```
void __bang_taylor3_cos(half *dst,
                        half *src,
                        half *aux1,
                        half *aux2,
                        int elem_count)
```

```
void __bang_taylor3_cos(float *dst,
                        float *src,
                        float *aux1,
                        float *aux2,
                        int elem_count)
```

Applies active with taylor3 (cos) operation on <src>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `aux1`: The auxiliary `__nram__` space 1.
- [in] `aux2`: The auxiliary `__nram__` space 2.
- [in] `elem_count`: Number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` , `<aux1>` and `<aux2>` must have the same size as `<src>`;
- `<src>` , `<aux1>` , `<aux2>` and `<dst>` must point to `__nram__` address space;
- `<dst>` , `<aux1>` , `<aux2>` and `<src>` cannot be overlapped;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src>`, `<aux1>`, `<aux2>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- The value of elements in `<src>` is in the range (-7.75, 7.75) radian; the average relative error is within 0.0005 and 0.00001 for `half` and `float` type respectively;
- Refer to the following tables for special value of `__bang_taylor3_cos` .

Table 3.23: Special Value of __bang_taylor3_cos(half)

Input	$-NaN$	$-INF$	$[-65504, -7.75]$	$[7.75, 65504]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0
Output on (m)tp_3xx or higher	+NaN	+NaN	+0.0	+0.0	+NaN	+NaN

Table 3.24: Special Value of __bang_taylor3_cos(float)

Input	$-NaN$	$-INF$	$[-3.4e38, -7.75]$	$[7.75, 3.4e38]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0
Output on (m)tp_3xx or higher	+NaN	+NaN	+0.0	+0.0	+NaN	+NaN

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.24 __bang_taylor3_sigmoid

```
void __bang_taylor3_sigmoid(half *dst,
                            half *src,
                            half *aux1,
                            half *aux2,
                            int elem_count)
```

```
void __bang_taylor3_sigmoid(float *dst,
                            float *src,
                            float *aux1,
                            float *aux2,
                            int elem_count)
```

Applies active with taylor3 (sigmoid) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The value of elements in <src> is in the range (-7.75, 7.75); the average relative error is within 0.0003 and 0.000001 for half and float type respectively;
- Refer to the following tables for special value of __bang_taylor3_sigmoid .

Table 3.25: Special Value of __bang_taylor3_sigmoid(half)

Input	$-NaN$	$-INF$	$[-65504, -7.75]$	$[7.75, 65504]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	5.96e-8	5.96e-8	5.96e-8	1	1	1
Output on (m)tp_3xx or higher	$+NaN$	$+NaN$	5.96e-8	1	$+NaN$	$+NaN$

Table 3.26: Special Value of __bang_taylor3_sigmoid(float)

Input	$-NaN$	$-INF$	$[-3.4e38, -7.75]$	$[7.75, 3.4e38]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	$1.4e-45$	$1.4e-45$	$1.4e-45$	0.99999994	0.99999994	0.99999994
Output on (m)tp_3xx or higher	$+NaN$	$+NaN$	$1.4e-45$	0.99999994	$+NaN$	$+NaN$

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.25 __bang_taylor3_sin

```
void __bang_taylor3_sin(half *dst,
                        half *src,
                        half *aux1,
                        half *aux2,
                        int elem_count)
```

```
void __bang_taylor3_sin(float *dst,
                        float *src,
                        float *aux1,
                        float *aux2,
                        int elem_count)
```

Applies active with taylor3 (sine) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>`, `<aux1>` and `<aux2>` must have the same size as `<src>`;
- `<src>`, `<aux1>`, `<aux2>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<aux1>`, `<aux2>` and `<src>` cannot be overlapped;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src>`, `<aux1>`, `<aux2>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- The value of elements in `<src>` is in the range (-7.75, 7.75) radian; the average relative error is within 0.0005 and 0.00001 for `half` and `float` type respectively;
- Refer to the following tables for special value of `__bang_taylor3_sin`.

Table 3.27: Special Value of `__bang_taylor3_sin(half)`

Input	<code>-NaN</code>	<code>-INF</code>	<code>[-65504, -7.75]</code>	<code>[7.75, 65504]</code>	<code>+INF</code>	<code>+NaN</code>
Output on <code>(m)tp_2xx</code> or earlier	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>
Output on <code>(m)tp_3xx</code> or higher	<code>+NaN</code>	<code>+NaN</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+NaN</code>	<code>+NaN</code>

Cambricon@155chb
Table 3.28: Special Value of `__bang_taylor3_sin(float)`

Input	<code>-NaN</code>	<code>-INF</code>	<code>[-3.4e38, -7.75]</code>	<code>[7.75, 3.4e38]</code>	<code>+INF</code>	<code>+NaN</code>
Output on <code>(m)tp_2xx</code> or earlier	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+0.0</code>
Output on <code>(m)tp_3xx</code> or higher	<code>+NaN</code>	<code>+NaN</code>	<code>+0.0</code>	<code>+0.0</code>	<code>+NaN</code>	<code>+NaN</code>

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.13.26 __bang_taylor3_softplus

```
void __bang_taylor3_softplus(half *dst,
                             half *src,
                             half *aux1,
                             half *aux2,
                             int elem_count)
```

```
void __bang_taylor3_softplus(float *dst,
                             float *src,
                             float *aux1,
                             float *aux2,
                             int elem_count)
```

Applies active with taylor3 (softplus) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

Cambricon@155chb

- <elem_count> must be greater than zero;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The value of elements in <src> is in the range (-7.75, 7.75) ; the average relative error is within 0.0003 and 0.000001 for half and float type respectively;
- Refer to the following tables for special value of __bang_taylor3_softplus .

Table 3.29: Special Value of __bang_taylor3_softplus(half)

Input	$-NaN$	$-INF$	$[-65504, -7.75]$	$[7.75, 65504]$	$+INF$	$+NaN$
Output on (m)tp_2x or earlier	0.000215292	0.000215292	0.000215292	$y = x$	65504	65504
Output on (m)tp_3x or higher	$+NaN$	$+NaN$	0.000215292	$y = x$	$+NaN$	$+NaN$

Table 3.30: Special Value of __bang_taylor3_softplus(float)

Input	$-NaN$	$-INF$	$[-3.4e38, -7.75]$	$[7.75, 3.4e38]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	0.0002153 2489	0.0002153 2489	0.0002153 2489	$y = x + 4.76837e-7$	3.4e38	3.4e38
Output on (m)tp_3xx or higher	$+NaN$	$+NaN$	0.0002153 2489	$y = x + 4.76837e-7$	$+NaN$	$+NaN$

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.27 __bang_taylor3_tanh

```
void __bang_taylor3_tanh(half *dst,
                         half *src,
                         half *aux1,
                         half *aux2,
                         int elem_count)
```

```
void __bang_taylor3_tanh(float *dst,
                         float *src,
                         float *aux1,
                         float *aux2,
                         int elem_count)
```

Applies active with taylor3 (tanh) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

Cambricon@155chb

- <elem_count> must be greater than zero;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The value of elements in <src> must be in the range (-7.75, 7.75) ; the average relative error is within 0.0003 and 0.00001 for half and float type respectively.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.28 __bang_taylor4_cos

```
void __bang_taylor4_cos(half *dst,
                        half *src,
                        half *aux1,
                        half *aux2,
                        int elem_count)
```

```
void __bang_taylor4_cos(float *dst,
                        float *src,
                        float *aux1,
                        float *aux2,
                        int elem_count)
```

Applies active with taylor4 (cos) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

Cambricon@155chb

- <elem_count> must be greater than zero;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The value of elements in <src> is in the range (-7.75, 7.75) radian; the average relative error is within 0.0005 and 0.000001 for half and float type respectively;
- Refer to the following tables for special value of __bang_taylor4_cos .

Table 3.31: Special Value of __bang_taylor4_cos(half)

Input	-NaN	-INF	[-65504, -7.75]	[7.75, 65504]	+INF	+NaN
Output on (m)tp_2xx or earlier	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0
Output on (m)tp_3xx or higher	+NaN	+NaN	+0.0	+0.0	+NaN	+NaN

Table 3.32: Special Value of __bang_taylor4_cos(float)

Input	$-NaN$	$-INF$	$[-3.4e38, -7.75]$	$[7.75, 3.4e38]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0
Output on (m)tp_3xx or higher	+NaN	+NaN	+0.0	+0.0	+NaN	+NaN

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.29 __bang_taylor4_sigmoid

```
void __bang_taylor4_sigmoid(half *dst,
                            half *src,
                            half *aux1,
                            half *aux2,
                            int elem_count)
```

```
void __bang_taylor4_sigmoid(float *dst,
                            float *src,
                            float *aux1,
                            float *aux2,
                            int elem_count)
```

Applies active with taylor4 (sigmoid) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- The value of elements in <src> is in the range (-7.75, 7.75); the average relative error is within 0.0003 and 0.0000001 for half and float type respectively;
- Refer to the following tables for special value of __bang_taylor4_sigmoid.

Table 3.33: Special Value of __bang_taylor4_sigmoid(half)

Input	<i>-NaN</i>	<i>-INF</i>	[<i>-65504, -7.75</i>]	[<i>7.75, 65504</i>]	<i>+INF</i>	<i>+NaN</i>
Output on (m)tp_2xx or earlier	5.96e-8	5.96e-8	5.96e-8	1	1	1
Output on (m)tp_3xx or higher	<i>+NaN</i>	<i>+NaN</i>	5.96e-8	1	<i>+NaN</i>	<i>+NaN</i>

Table 3.34: Special Value of __bang_taylor4_sigmoid(float)

Input	<i>-NaN</i>	<i>-INF</i>	[<i>-3.4e38, -7.75</i>]	[<i>7.75, 3.4e38</i>]	<i>+INF</i>	<i>+NaN</i>
Output on (m)tp_2xx or earlier	1.4e-45	1.4e-45	1.4e-45	0.99999994	0.99999994	0.99999994
Output on (m)tp_3xx or higher	<i>+NaN</i>	<i>+NaN</i>	1.4e-45	0.99999994	<i>+NaN</i>	<i>+NaN</i>

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.30 __bang_taylor4_sin

```
void __bang_taylor4_sin(half *dst,
                        half *src,
                        half *aux1,
                        half *aux2,
                        int elem_count)
```

```
void __bang_taylor4_sin(float *dst,
                        float *src,
                        float *aux1,
                        float *aux2,
                        int elem_count)
```

Applies active with taylor4 (sine) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

Cambricon@155chb

- <elem_count> must be greater than zero;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The value of elements in <src> is in the range (-7.75, 7.75) radian; the average relative error is within 0.0005 and 0.000001 for half and float type respectively;
- Refer to the following tables for special value of __bang_taylor4_sin .

Table 3.35: Special Value of __bang_taylor4_sin(half)

Input	-NaN	-INF	[-65504, -7.75]	[7.75, 65504]	+INF	+NaN
Output on (m)tp_2xx or earlier	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0
Output on (m)tp_3xx or higher	+NaN	+NaN	+0.0	+0.0	+NaN	+NaN

Table 3.36: Special Value of __bang_taylor4_sin(float)

Input	$-NaN$	$-INF$	$[-3.4e38, -7.75]$	$[7.75, 3.4e38]$	$+INF$	$+NaN$
Output on (m)tp_2xx or earlier	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0
Output on (m)tp_3xx or higher	+NaN	+NaN	+0.0	+0.0	+NaN	+NaN

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.31 __bang_taylor4_softplus

```
void __bang_taylor4_softplus(half *dst,
                             half *src,
                             half *aux1,
                             half *aux2,
                             int elem_count)
```

```
void __bang_taylor4_softplus(float *dst,
                             float *src,
                             float *aux1,
                             float *aux2,
                             int elem_count)
```

Applies active with taylor4 (softplus) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>`, `<aux1>` and `<aux2>` must have the same size as `<src>`;
- `<src>`, `<aux1>`, `<aux2>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<aux1>`, `<aux2>` and `<src>` cannot be overlapped;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src>`, `<aux1>`, `<aux2>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- The value of elements in `<src>` is in the range (-7.75, 7.75); the average relative error is within 0.0003 and 0.0000001 for `half` and `float` type respectively;
- Refer to the following tables for special value of `__bang_taylor4_softplus`.

Table 3.37: Special Value of `__bang_taylor4_softplus(half)`

Input	<code>-NaN</code>	<code>-INF</code>	<code>[-65504, -7.75]</code>	<code>[7.75, 65504]</code>	<code>+INF</code>	<code>+NaN</code>
Output on <code>(m)tp_2x</code> or earlier	0.000215292	0.000215292	0.000215292	$y = x$	65504	65504
Output on <code>(m)tp_3x</code> or higher	<code>+NaN</code>	<code>+NaN</code>	0.000215292	$y = x$	<code>+NaN</code>	<code>+NaN</code>

Table 3.38: Special Value of `__bang_taylor4_softplus(float)`

Input	<code>-NaN</code>	<code>-INF</code>	<code>[-3.4e38, -7.75]</code>	<code>[7.75, 3.4e38]</code>	<code>+INF</code>	<code>+NaN</code>
Output on <code>(m)tp_2xx</code> or earlier	0.0002153 2489	0.0002153 2489	0.0002153 2489	$y = x + 4.76837e-7$	3.4e38	3.4e38
Output on <code>(m)tp_3xx</code> or higher	<code>+NaN</code>	<code>+NaN</code>	0.0002153 2489	$y = x + 4.76837e-7$	<code>+NaN</code>	<code>+NaN</code>

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;

- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.13.32 __bang_taylor4_tanh

```
void __bang_taylor4_tanh(half *dst,
                         half *src,
                         half *aux1,
                         half *aux2,
                         int elem_count)
```

```
void __bang_taylor4_tanh(float *dst,
                         float *src,
                         float *aux1,
                         float *aux2,
                         int elem_count)
```

Applies active with taylor4 (tanh) operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] aux1: The auxiliary __nram__ space 1.
- [in] aux2: The auxiliary __nram__ space 2.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst>, <aux1> and <aux2> must have the same size as <src>;
- <src>, <aux1>, <aux2> and <dst> must point to __nram__ address space;
- <dst>, <aux1>, <aux2> and <src> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src>, <aux1>, <aux2> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The value of elements in <src> must be in the range (-7.75, 7.75); the average relative error is within 0.0003 and 0.000001 for half and float type respectively.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.14 Vector Bitwise Functions

3.14.1 __bang_band

```
void __bang_band(char *dst,
                  char *src0,
                  char *src1,
                  int elem_count)
```

Applies bit-wise AND operation on two vectors.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src0>;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(char* c, char* a, char* b) {
    __nram__ char a_tmp[DATA_SIZE];
    __nram__ char c_tmp[DATA_SIZE];
    __nram__ char b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE, GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE, GDRAM2NRAM);
    __bang_band(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE, NRAM2GDRAM);
}
```

3.14.2 __bang_band_scalar

```
void __bang_band_scalar(half *dst,
                        half *src,
                        half value,
                        int elem_count)

void __bang_band_scalar(float *dst,
                        float *src,
                        float value,
                        int elem_count)

void __bang_band_scalar(bfloat16_t *dst,
                        bfloat16_t *src,
                        bfloat16_t value,
                        int elem_count)

void __bang_band_scalar(int *dst,
                        int *src,
                        int value,
                        int elem_count)

void __bang_band_scalar(short *dst,
                        short *src,
                        short value,
                        int elem_count)

void __bang_band_scalar(char *dst,
                        char *src,
                        char value,
                        int elem_count)
```

This function performs AND operation between `<elem_count>` elements of `<src>` and `<value>` bit-wise and saves the result in `<dst>`. If both the element of `<src>` and `<value>` are non-zero, the bit of result is 1. Otherwise, the bit of result is 0.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;

- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(int* c, int* a, intt b) {
    __nram__ int a_tmp[DATA_SIZE];
    __nram__ int c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(int), GDRAM2NRAM);
    __bang_band_scalar(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(int), NRAM2GDRAM);
}
```

3.14.3 __bang_bnot

Cambricon@155chb

```
void __bang_bnot(char *dst,
                  char *src,
                  int elem_count)
```

Applies bit-wise NOT operation on a vector.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.14.4 __bang_bor

```
void __bang_bor(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)
```

Applies bit-wise OR operation on two vectors.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of first source vector.
- [in] `src1`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` can be overlapped with `<src0>`;
- `<src0>, <src1> and <dst>` must point to `__nram__` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>, <src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.14.5 __bang_bor_scalar

```
void __bang_bor_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_bor_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

```
void __bang_bor_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_bor_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_bor_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_bor_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

This function performs OR operation between `<elem_count>` elements of `<src>` and `<value>` bit-wise and saves the result in `<dst>`. If either the element of `<src>` or `<value>` is non-zero, the bit of result is 1. Otherwise, the bit of result is 0.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of `__bang_band_scalar` for more details.

3.14.6 `__bang_bxor`

```
void __bang_bxor(char *dst,
                  char *src0,
                  char *src1,
                  int elem_count)
```

Applies bit-wise XOR operation on two vectors.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of first source vector.
- [in] `src1`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` can be overlapped with `<src0>`;
- `<src0>, <src1> and <dst>` must point to `__nram__` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>, <src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.14.7 `__bang_bxor_scalar`

```
void __bang_bxor_scalar(half *dst,
                       half *src,
                       half value,
                       int elem_count)
```

```
void __bang_bxor_scalar(float *dst,
                        float *src,
                        float value,
                        int elem_count)
```

```
void __bang_bxor_scalar(bfloat16_t *dst,
                        bfloat16_t *src,
                        bfloat16_t value,
                        int elem_count)
```

```
void __bang_bxor_scalar(short *dst,
                        short *src,
                        short value,
                        int elem_count)
```

```
void __bang_bxor_scalar(int *dst,
                        int *src,
                        int value,
                        int elem_count)
```

```
void __bang_bxor_scalar(char *dst,
                        char *src,
                        char value,
                        int elem_count)
```

This function performs XOR operation between `<elem_count>` elements of `<src>` and `<value>` bit-wise and saves the result in `<dst>`. If the element of `<src>` is zero and `<value>` is non-zero, or the element of `<src>` is non-zero and `<value>` is zero, the bit of result is 1. Otherwise, the bit of result is 0.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of `__bang_band_scalar` for more details.

3.14.8 __bang_cycle_band

```
void __bang_cycle_band(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts. Each element in each part applies bit-wise AND operation with the corresponding element in `<seg>`. The result is assigned to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Cambricon@155chb

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_elem_count> * sizeof(char)` and `<seg_elem_count> * sizeof(char)` must be divisible by 128 on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.14.9 __bang_cycle_bor

```
void __bang_cycle_bor(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts. Each element in each part applies bit-wise OR operation with the corresponding element in `<seg>`. The result is assigned to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_elem_count> * sizeof(char)` and `<seg_elem_count> * sizeof(char)` must be divisible by 128 on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.14.10 __bang_cycle_bxor

```
void __bang_cycle_bxor(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts. Each element in each part applies bit-wise XOR operation with the corresponding element in `<seg>`. The result is assigned to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_elem_count> * sizeof(char)` and `<seg_elem_count> * sizeof(char)` must be divisible by 128 on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.15 Vector Comparison Functions

3.15.1 __bang_argmax

```
void __bang_argmax(half *dst,
                    half *src,
                    int elem_count)
```

```
void __bang_argmax(bfloat16_t *dst,
                    bfloat16_t *src,
                    int elem_count)
```

```
void __bang_argmax(float *dst,
                    float *src,
                    int elem_count)
```

Finds the maximum value and its corresponding index in <src> vector. The result is composed of two parts. The first part is the maximum value of corresponding type, and the second part is the index of the first maximum value position in <src> vector, whose data type is `unsigned int`. The maximum value and index are stored continuously in <dst>. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define DATA_NUM 90

__mlu_entry__ void kernel_max(float* dst, float* src) {
    __nram__ float dst_nram[DATA_NUM];
    __nram__ float src_nram[DATA_NUM];
    __memcpy(src_nram, src, DATA_NUM * sizeof(float), GDRAM2NRAM);
    __bang_argmax(dst_nram, src_nram, DATA_NUM);
    __memcpy(dst, dst_nram, DATA_NUM * sizeof(float), NRAM2GDRAM);
}
```

3.15.2 __bang_argmin

```
void __bang_argmin(half *dst,
                    half *src,
                    int elem_count)

void __bang_argmin(bfloat16_t *dst,
                    bfloat16_t *src,
                    int elem_count)

void __bang_argmin(float *dst,
                    float *src,
                    int elem_count)
```

Finds the minimum value and its corresponding index in <src> vector. The result is composed of two parts. The first part is the minimum value of corresponding data type, and the second part is the index of the first minimum value position in <src> vector, whose data type is `unsigned int`. The minimum value and index are stored continuously. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define DATA_NUM 90

__mlu_entry__ void kernel_min(float* dst, float* src) {
    __nram__ float dst_nram[DATA_NUM];
    __nram__ float src_nram[DATA_NUM];
```

```

__memcpy(src_nram, src, DATA_NUM * sizeof(float), GDRAM2NRAM);
__bang_argmin(dst_nram, src_nram, DATA_NUM);
__memcpy(dst, dst_nram, DATA_NUM * sizeof(float), NRAM2GDRAM);
}

```

3.15.3 __bang_cycle_eq

```
void __bang_cycle_eq(int *dst,
                     int *src,
                     int *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_eq(short *dst,
                     short *src,
                     short *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_eq(char *dst,
                     char *src,
                     char *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_eq(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_eq(half *dst,
                     half *src,
                     half *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_eq(float *dst,
                     float *src,
                     float *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

Divides <src> into <src_elem_count> / <seg_elem_count> parts, then judges whether each element in each part and the corresponding element in <seg> are equal. The result is

assigned to <dst>. If the element of <src> is equal to <value>, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of <src>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <dst> can be overlapped with <src>;
- <src_elem_count> must be divisible by <seg_elem_count>;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <src>, <seg> and <dst> must point to __nram__ address space;
- int, short and char are supported on (m)tp_3xx or higher;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_elem_count> * sizeof(type) and <seg_elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define N1 128
#define N2 32640

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[N1];
    __nram__ float b_tmp[N2];
    __nram__ float c_tmp[N2];
    __memcpy(a_tmp, a, N1 * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, N2 * sizeof(float), GDRAM2NRAM);
    __bang_cycle_eq(c_tmp, b_tmp, a_tmp, N2, N1);
    __memcpy(c, c_tmp, N2 * sizeof(float), NRAM2GDRAM);
}
```

3.15.4 __bang_cycle_equ

```
void __bang_cycle_equ(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_equ(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_equ(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts, then judges whether each element in each part and the corresponding element in `<seg>` are unordered or equal. The result is assigned to `<dst>`. If the element of `<src>` is unordered or equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define N1 128
#define N2 32640

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[N1];
    __nram__ float b_tmp[N2];
    __nram__ float c_tmp[N2];
    __memcpy(a_tmp, a, N1 * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, N2 * sizeof(float), GDRAM2NRAM);
    __bang_cycle_equ(c_tmp, b_tmp, a_tmp, N2, N1);
    __memcpy(c, c_tmp, N2 * sizeof(float), NRAM2GDRAM);
}
```

3.15.5 __bang_cycle_ge

```
void __bang_cycle_ge(int *dst,
                     int *src,
                     int *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(short *dst,
                     short *src,
                     short *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(char *dst,
                     char *src,
                     char *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(unsigned int *dst,
                     unsigned int *src,
                     unsigned int *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(unsigned short *dst,
                     unsigned short *src,
                     unsigned short *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(unsigned char *dst,
                     unsigned char *src,
                     unsigned char *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(half *dst,
                     half *src,
                     half *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ge(float *dst,
                     float *src,
                     float *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts, then judges whether each element in each part is greater than or equal to the corresponding element in `<seg>`. The result is assigned to `<dst>`. If the element of `<src>` is greater than or equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- <dst> can be overlapped with <src>;
- <src>, <seg> and <dst> must point to __nram__ address space;
- <src_elem_count> must be divisible by <seg_elem_count>;
- bfloat16_t is supported on (m)tp_5xx or higher;
- int, short and char are supported on (m)tp_3xx or higher;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_elem_count> * sizeof(type) and <seg_elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.15.6 __bang_cycle_geu

```
void __bang_cycle_geu(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_geu(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_geu(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides <src> into <src_elem_count> / <seg_elem_count> parts, then judges whether each element in each part is unordered or greater than or equal to the corresponding element in <seg>. The result is assigned to <dst>. If the element of <src> is unordered or greater than or equal to <value>, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of <src>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] *dst*: The address of destination vector.
- [in] *src*: The address of first source vector.
- [in] *seg*: The address of second source vector.
- [in] *src_elem_count*: The number of elements in <*src*> vector.
- [in] *seg_elem_count*: The number of elements in <*seg*> vector.

Return

- void.

Remark

- <*dst*> can be overlapped with <*src*>;
- <*src_elem_count*> must be divisible by <*seg_elem_count*>;
- <*src*>, <*seg*> and <*dst*> must point to __nram__ address space;
- <*src_elem_count*> and <*seg_elem_count*> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- See the example of [__bang_cycle_eq](#) for more details.

3.15.7 __bang_cycle_gt

```
void __bang_cycle_gt(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_gt(half *dst,
                     half *src,
                     half *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_gt(float *dst,
                     float *src,
                     float *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

Divides <*src*> into <*src_elem_countseg_elem_count*> parts, then judges whether each element in each part is greater than the corresponding element in <*seg*>. The result is assigned to <*dst*>. If the element of <*src*> is greater than <*value*>, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of <*src*>. See the table

Floating Point Calculation of Stream and Scalar Comparison Functions for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <dst> can be overlapped with <src>;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <src>, <seg> and <dst> must point to __nram__ address space;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_elem_count> * sizeof(type) and <seg_elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: --BANG_ARCH-- >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.15.8 __bang_cycle_gtu

```
void __bang_cycle_gtu(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_gtu(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_gtu(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts, then judges whether each element in each part is unordered or greater than the corresponding element in `<seg>`. The result is assigned to `<dst>`. If the element of `<src>` is unordered or greater than `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_cycle_eq](#) for more details.

3.15.9 __bang_cycle_le

```
void __bang_cycle_le(half *dst,
                     half *src,
                     half *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_le(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_le(float *dst,
                     float *src,
                     float *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts, then judges whether each element in each part is less than or equal to the corresponding element in `<seg>`. The result is assigned to `<dst>`. If the element of `<src>` is less than or equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.15.10 __bang_cycle_leu

```
void __bang_cycle_leu(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_leu(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_leu(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts, then judges whether each element in each part is unordered or less than or equal to the corresponding element in `<seg>`. The result is assigned to `<dst>`. If the element of `<src>` is unordered or less than or equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_cycle_eq](#) for more details.

3.15.11 __bang_cycle_lt

```
void __bang_cycle_lt(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_lt(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_lt(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_lt(unsigned int *dst,
                      unsigned int *src,
                      unsigned int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_lt(unsigned short *dst,
                      unsigned short *src,
                      unsigned short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_lt(unsigned char *dst,
                      unsigned char *src,
                      unsigned char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_lt(half *dst,
                     half *src,
                     half *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_lt(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_lt(float *dst,
                     float *src,
                     float *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts, then judges whether each element in each part is less than the corresponding element in `<seg>`. The result is assigned to `<dst>`. If the element of `<src>` is less than `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;

- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.15.12 __bang_cycle_ltu

```
void __bang_cycle_ltu(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_ltu(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_ltu(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts, then judges whether each element in each part is unordered or less than the corresponding element in `<seg>`. The result is assigned to `<dst>`. If the element of `<src>` is unordered or less than `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- See the example of [__bang_cycle_eq](#) for more details.

3.15.13 __bang_cycle_maxequal

```
void __bang_cycle_maxequal(int *dst,
                           int *src,
                           int *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(short *dst,
                           short *src,
                           short *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(char *dst,
                           char *src,
                           char *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(unsigned int *dst,
                           unsigned int *src,
                           unsigned int *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(unsigned short *dst,
                           unsigned short *src,
                           unsigned short *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(unsigned char *dst,
                           unsigned char *src,
                           unsigned char *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(half *dst,
                           half *src,
                           half *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(bfloat16_t *dst,
                           bfloat16_t *src,
                           bfloat16_t *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maxequal(float *dst,
                           float *src,
                           float *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

Cambricon@155chb

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts. Each element in each part applies maxequal operation (select the maximum value) with the corresponding element in `<seg>`. The result is assigned to `<dst>`. If one of the elements being compared is NAN, then the value in `<src>` is returned.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `int, short` and `char` are supported on `(m)tp_3xx` or higher;
- `<src>, <seg>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>, <seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;

- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define N1 128
#define N2 32640

__mlu_entry__ void kernel(float *c, float *a, float *b) {
    __nram__ float a_tmp[N1];
    __nram__ float b_tmp[N2];
    __nram__ float c_tmp[N2];
    __memcpy(a_tmp, a, N1 * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, N2 * sizeof(float), GDRAM2NRAM);
    __bang_cycle_maxequal(c_tmp, b_tmp, a_tmp, N2, N1);
    __memcpy(c, c_tmp, N2 * sizeof(float), NRAM2GDRAM);
}
```

Cambricon@155chb

3.15.14 __bang_cycle_maximum

```
void __bang_cycle_maximum(float *dst,
                           float *src,
                           float *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maximum(half *dst,
                           half *src,
                           half *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_maximum(bfloat16_t *dst,
                           bfloat16_t *src,
                           bfloat16_t *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

Divides <src> into <src_elem_count> / <seg_elem_count> parts. Each element in each part applies max operation (select the maximum value) with the corresponding element in <seg>.

The result is assigned to <dst>. If both of the elements being compared are NaN, then NaN is returned.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <src_elem_count> and <seg_elem_count> must be greater than zero;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src>, <seg> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- None.

Cambricon@155chb

3.15.15 __bang_cycle_minequal

```
void __bang_cycle_minequal(int *dst,
                           int *src,
                           int *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(short *dst,
                           short *src,
                           short *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(char *dst,
                           char *src,
                           char *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(unsigned int *dst,
                            unsigned int *src,
                            unsigned int *seg,
                            int src_elem_count,
                            int seg_elem_count)
```

```
void __bang_cycle_minequal(unsigned short *dst,
                           unsigned short *src,
                           unsigned short *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(unsigned char *dst,
                           unsigned char *src,
                           unsigned char *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(half *dst,
                           half *src,
                           half *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(bfloat16_t *dst,
                           bfloat16_t *src,
                           bfloat16_t *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minequal(float *dst,
                           float *src,
                           float *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts. Each element in each part applies `minequal` operation (select the minimum value) with the corresponding element in `<seg>`. The result is assigned to `<dst>`. If one of the elements being compared is NAN, then the value in `<src>` is returned.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.

- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `_nram__` address space;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.15.16 `__bang_cycle_minimum`

```
void __bang_cycle_minimum(float *dst,
                           float *src,
                           float *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minimum(half *dst,
                           half *src,
                           half *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

```
void __bang_cycle_minimum(bfloat16_t *dst,
                           bfloat16_t *src,
                           bfloat16_t *seg,
                           int src_elem_count,
                           int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts. Each element in each part applies min operation (select the minimum value) with the corresponding element in `<seg>`. The result is assigned to `<dst>`. If both of the elements being compared are NaN, then NaN is

returned.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <src_elem_count> and <seg_elem_count> must be greater than zero;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src>, <seg> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- None.

Cambricon@155chb

3.15.17 __bang_cycle_nan_maximum

```
void __bang_cycle_nan_maximum(float *dst,
                               float *src,
                               float *seg,
                               int src_elem_count,
                               int seg_elem_count)
```

```
void __bang_cycle_nan_maximum(bfloat16_t *dst,
                               bfloat16_t *src,
                               bfloat16_t *seg,
                               int src_elem_count,
                               int seg_elem_count)
```

```
void __bang_cycle_nan_maximum(half *dst,
                               half *src,
                               half *seg,
                               int src_elem_count,
                               int seg_elem_count)
```

Divides <src> into <src_elem_count> / <seg_elem_count> parts. Each element in each part applies max operation (select the maximum value) with the corresponding element in <seg>.

The result is assigned to <dst>. If one of the elements being compared is a NaN, then that element is returned.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <src_elem_count> and <seg_elem_count> must be greater than zero;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src>, <seg> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.15.18 __bang_cycle_nan_minimum

```
void __bang_cycle_nan_minimum(float *dst,
                               float *src,
                               float *seg,
                               int src_elem_count,
                               int seg_elem_count)
```

```
void __bang_cycle_nan_minimum(bfloat16_t *dst,
                               bfloat16_t *src,
                               bfloat16_t *seg,
                               int src_elem_count,
                               int seg_elem_count)
```

```
void __bang_cycle_nan_minimum(half *dst,
                               half *src,
                               half *seg,
                               int src_elem_count,
                               int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts. Each element in each part applies min operation (select the minimum value) with the corresponding element in `<seg>`. The result is assigned to `<dst>`. If one of the elements being compared is a NaN, then that element is returned.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- `<dst>` can be overlapped with `<src>`;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

- None.

3.15.19 __bang_cycle_ne

```
void __bang_cycle_ne(int *dst,
                     int *src,
                     int *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ne(short *dst,
                     short *src,
                     short *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_ne(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_ne(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_ne(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_ne(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Cambricon@155chb

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts, then judges whether each element in each part and the corresponding element in `<seg>` are NOT equal. The result is assigned to `<dst>`. If the element of `<src>` is not equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `int, short` and `char` are supported on `(m)tp_3xx` or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;

- <src>, <seg> and <dst> must point to __nram__ address space;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.15.20 __bang_cycle_neu

```
void __bang_cycle_neu(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_neu(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_neu(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides <src> into <src_elem_count> / <seg_elem_count> parts, then judges whether each element in each part and the corresponding element in <seg> are unordered or not equal. The result is assigned to <dst>. If the element of <src> is unordered or not equal to <value>, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of <src>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <dst> can be overlapped with <src>;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src>, <seg> and <dst> must point to __nram__ address space;
- <src_elem_count> and <seg_elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- See the example of [__bang_cycle_eq](#) for more details.

3.15.21 __bang_eq

```
void __bang_eq(int *dst,
               int *src0,
               int *src1,
               int elem_count)
```

```
void __bang_eq(short *dst,
               short *src0,
               short *src1,
               int elem_count)
```

```
void __bang_eq(char *dst,
               char *src0,
               char *src1,
               int elem_count)
```

```
void __bang_eq(half *dst,
               half *src0,
               half *src1,
               int elem_count)
```

```
void __bang_eq(bfloat16_t *dst,
               bfloat16_t *src0,
               bfloat16_t *src1,
               int elem_count)
```

```
void __bang_eq(float *dst,
               float *src0,
               float *src1,
               int elem_count)
```

This function performs equality comparison with `<src0>` and `<src1>` and saves the result in `<dst>`. If the element of `<src0>` and the element of `<src1>` are equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_eq(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.22 __bang_eq_bitindex

```
void __bang_eq_bitindex(bfloat16_t *dst,
                       bfloat16_t *src0,
                       bfloat16_t *src1,
                       int elem_count)
```

```
void __bang_eq_bitindex(half *dst,
                       half *src0,
                       half *src1,
                       int elem_count)
```

```
void __bang_eq_bitindex(float *dst,
                       float *src0,
                       float *src1,
                       int elem_count)
```

This function performs equality comparison with `<src0>` and `<src1>` element-wisely. If the two elements are equal, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information. The comparison process is illustrated in the figure below, the comparison result between the first element `<element0>` of `<src0>` and the first element `<element0>` of `<src1>` will be saved in `<bit0>` of `<dst>`. The other elements do the same comparison in turn.

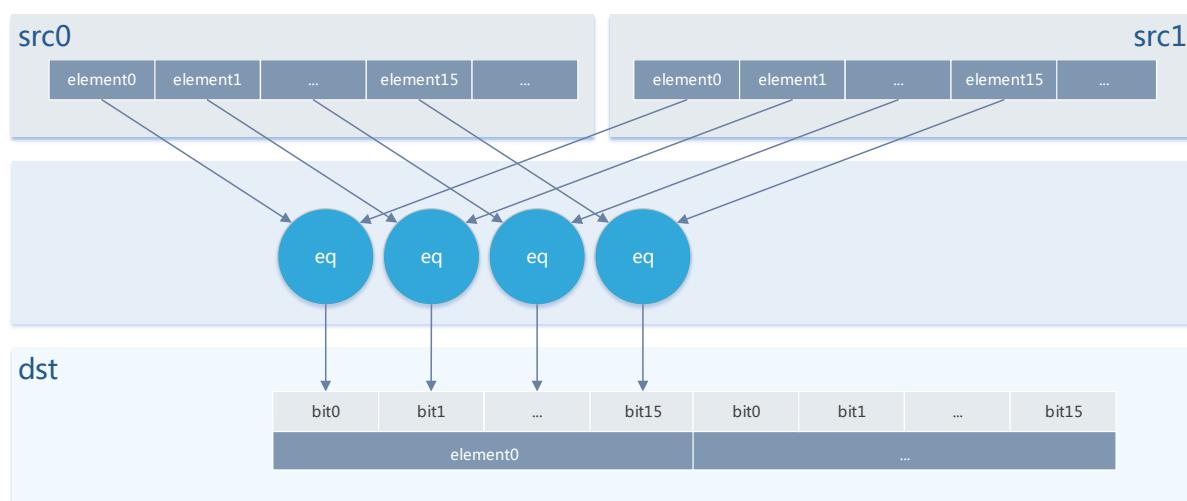


Fig. 3.6: The Calculation Process of Half Type `__bang_eq_bitindex`

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <elem_count> must be divisible by 512 on (m)tp_2xx;
- <elem_count> must be divisible by 8 on (m)tp_3xx or higher;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_eq_bitindex(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.23 __bang_eq_scalar

```
void __bang_eq_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_eq_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```

void __bang_eq_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)

void __bang_eq_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_eq_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_eq_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)

```

This function performs equality comparison with `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. If the element of `<src>` and `<value>` are equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero;
- `bfloat16_t` is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_eq_scalar(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.24 __bang_eq

```
void __bang_eq(half *dst,
               half *src0,
               half *src1,
               int elem_count)
```

```
void __bang_eq(bfloat16_t *dst,
               bfloat16_t *src0,
               bfloat16_t *src1,
               int elem_count)
```

```
void __bang_eq(float *dst,
               float *src0,
               float *src1,
               int elem_count)
```

This function performs unordered or equality comparison with `<src0>` and `<src1>` and saves the result in `<dst>`. If the element of `<src0>` and the element of `<src1>` are unordered or equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_equ(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.25 __bang_equ_bitindex

```
void __bang_equ_bitindex(bfloat16_t*dst,
                        bfloat16_t*src0,
                        bfloat16_t*src1,
                        int elem_count)
```

```
void __bang_equ_bitindex(half*dst,
                        half*src0,
                        half*src1,
                        int elem_count)
```

```
void __bang_equ_bitindex(float*dst,
                        float*src0,
                        float*src1,
                        int elem_count)
```

This function performs unordered or equality comparison with `<src0>` and `<src1>` element-wisely. If the two elements are unordered or equal, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.

- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero and divisible by 8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_equ_bitindex(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.26 __bang_equ_scalar

```
void __bang_equ_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_equ_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_equ_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function performs unordered or equality comparison with <elem_count> elements of

`<src>` and `<value>` and saves the result in `<dst>`. If the element of `<src>` and `<value>` are unordered or equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

Cambricon@155chb

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_equ_scalar(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.27 `__bang_ge`

```
void __bang_ge(int *dst,
               int *src0,
               int *src1,
               int elem_count)
```

```
void __bang_ge(short *dst,
                short *src0,
                short *src1,
                int elem_count)
```

```
void __bang_ge(char *dst,
                char *src0,
                char *src1,
                int elem_count)
```

```
void __bang_ge(unsigned int *dst,
                unsigned int *src0,
                unsigned int *src1,
                int elem_count)
```

```
void __bang_ge(unsigned short *dst,
                unsigned short *src0,
                unsigned short *src1,
                int elem_count)
```

```
void __bang_ge(unsigned char *dst,
                unsigned char *src0,
                unsigned char *src1,
                int elem_count)
```

```
void __bang_ge(bfloat16_t *dst,
                bfloat16_t *src0,
                bfloat16_t *src1,
                int elem_count)
```

```
void __bang_ge(half *dst,
                half *src0,
                half *src1,
                int elem_count)
```

```
void __bang_ge(float *dst,
                float *src0,
                float *src1,
                int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is greater than or equal to that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is greater than or equal to the element of `<src1>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first vector.
- [in] src1: The address of the second vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- int, short and char are supported on (m)tp_3xx or higher;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_eq](#) for more details.

Cambricon@155chb

3.15.28 __bang_ge_bitindex

```
void __bang_ge_bitindex(bfloat16_t *dst,
                       bfloat16_t *src0,
                       bfloat16_t *src1,
                       int elem_count)
```

```
void __bang_ge_bitindex(half *dst,
                       half *src0,
                       half *src1,
                       int elem_count)
```

```
void __bang_ge_bitindex(float *dst,
                       float *src0,
                       float *src1,
                       int elem_count)
```

This function compares with <elem_count> elements in <src0> and <src1> element-wisely, if the element of <src0> is greater than or equal to that of <src1>, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in <dst>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information. The comparison process is illustrated in the figure below, the comparison result between the first

element <element0> of <src0> and the first element <element0> of <src1> will be saved in <bit0> of <dst>. The other elements do the same comparison in turn.

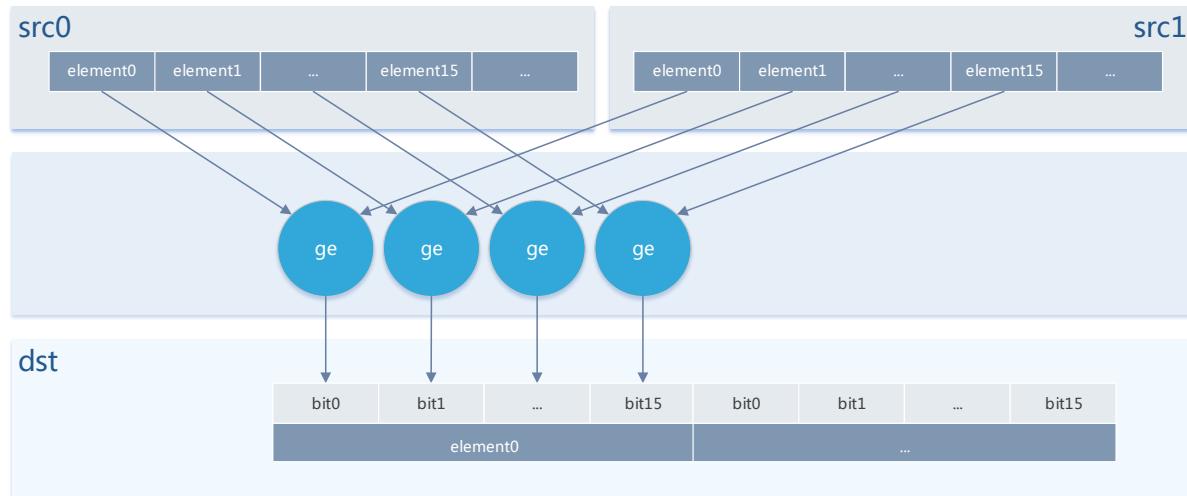


Fig. 3.7: The Calculation Process of Half Type `__bang_ge_bitindex`

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<elem_count>` must be divisible by 512 on `(m)tp_2xx`;
- `<elem_count>` must be divisible by 8 on `(m)tp_3xx` or higher;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_eq_bitindex` for more details.

3.15.29 __bang_ge_scalar

```
void __bang_ge_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_ge_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_ge_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_ge_scalar(unsigned int *dst,
                      unsigned int *src,
                      unsigned int value,
                      int elem_count)
```

```
void __bang_ge_scalar(unsigned short *dst,
                      unsigned short *src,
                      unsigned short value,
                      int elem_count)
```

```
void __bang_ge_scalar(unsigned char *dst,
                      unsigned char *src,
                      unsigned char value,
                      int elem_count)
```

```
void __bang_ge_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_ge_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_ge_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are greater than or equal to `<value>` and saves the result in `<dst>`. If the element of `<src>` is greater than or equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src>` can be overlapped with `<dst>`;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_ge_scalar(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.30 __bang_geu

```
void __bang_geu(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_geu(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_geu(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is unordered or greater than or equal to that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is unordered or greater than or equal to the element of `<src1>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first vector.
- [in] `src1`: The address of the second vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ](#) for more details.

3.15.31 __bang_geu_bitindex

```
void __bang_geu_bitindex(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_geu_bitindex(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_geu_bitindex(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely, if the element of `<src0>` is unoredered or greater than or equal to that of `<src1>`, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero and divisible by 8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ_bitindex](#) for more details.

3.15.32 __bang_geu_scalar

```
void __bang_geu_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_geu_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_geu_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are unordered or greater than or equal to `<value>` and saves the result in `<dst>`. If the element of `<src>` is unordered or greater than or equal to `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

Cambricon@155chb

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ_scalar](#) for more details.

3.15.33 __bang_gt

```
void __bang_gt(half *dst,
                half *src0,
                half *src1,
                int elem_count)

void __bang_gt(bfloat16_t *dst,
                bfloat16_t *src0,
                bfloat16_t *src1,
                int elem_count)

void __bang_gt(float *dst,
                float *src0,
                float *src1,
                int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is greater than that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is greater than the element of `<src1>`, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first vector.
- [in] `src1`: The address of the second vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_eq](#) for more details.

3.15.34 __bang_gt_bitindex

```
void __bang_gt_bitindex(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_gt_bitindex(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_gt_bitindex(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely, if the element of `<src0>` is greater than that of `<src1>`, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information. The comparison process is illustrated in the figure below, the comparison result between the first element `<element0>` of `<src0>` and the first element `<element0>` of `<src1>` will be saved in `<bit0>` of `<dst>`. The other elements do the same comparison in turn.

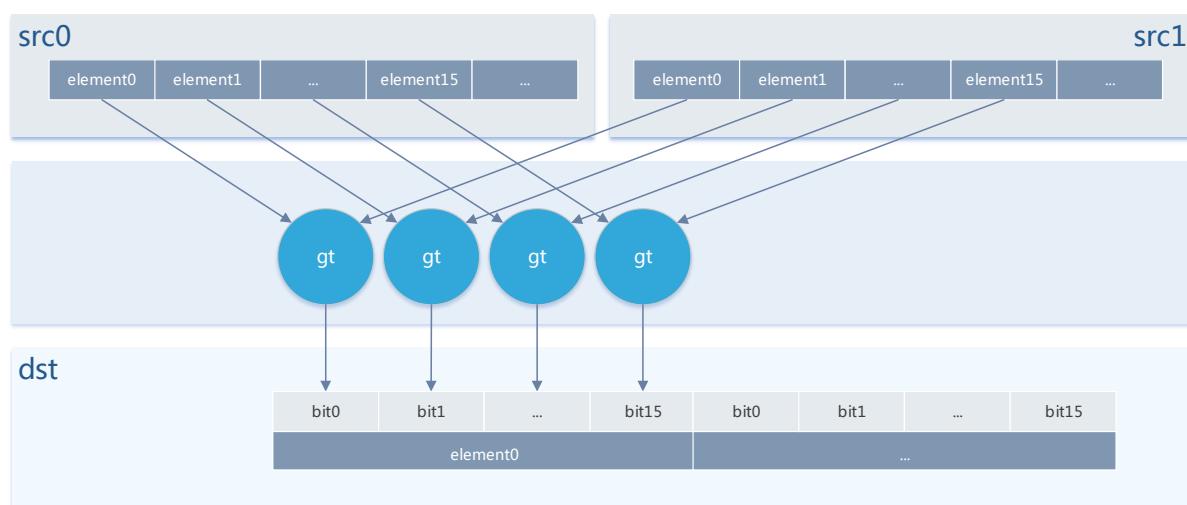


Fig. 3.8: The Calculation Process of Half Type `__bang_gt_bitindex`

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <elem_count> must be divisible by 512 on (m)tp_2xx;
- <elem_count> must be divisible by 8 on (m)tp_3xx or higher;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_eq_bitindex](#) for more details.

3.15.35 __bang_gt_scalar

```
void __bang_gt_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_gt_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_gt_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares <elem_count> elements in <src> with <value> to determine whether the elements are greater than <value> and saves the result in <dst>. If the element of <src> is greater than <value>, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of <src>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.

- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> must be greater than zero;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_eq_scalar](#) for more details.

3.15.36 __bang_gtu

```
void __bang_gtu(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_gtu(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_gtu(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function compares with <elem_count> elements in <src0> and <src1> element-wisely to determine whether the element in <src0> is unordered or greater than that in <src1> and saves the result in <dst>. If the element of <src0> is unordered or greater than the element of <src1>, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of <src0> and <src1>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first vector.
- [in] src1: The address of the second vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- See the example of [__bang_equ](#) for more details.

3.15.37 __bang_gtu_bitindex

```
void __bang_gtu_bitindex(bfloat16_t *dst,
                         bfloat16_t *src0,
                         bfloat16_t *src1,
                         int elem_count)
```

```
void __bang_gtu_bitindex(half *dst,
                         half *src0,
                         half *src1,
                         int elem_count)
```

```
void __bang_gtu_bitindex(float *dst,
                         float *src0,
                         float *src1,
                         int elem_count)
```

This function compares with <elem_count> elements in <src0> and <src1> element-wisely, if the element of <src0> is unordered or greater than that of <src1>, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in <dst>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;

- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero and divisible by 8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- See the example of [__bang_equ_bitindex](#) for more details.

3.15.38 __bang_gtu_scalar

```
void __bang_gtu_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_gtu_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_gtu_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares <elem_count> elements in <src> with <value> to determine whether the elements are unordered or greater than <value> and saves the result in <dst>. If the element of <src> is unordered or greater than <value>, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of <src>. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

- See the example of [__bang_equ_scalar](#) for more details.

3.15.39 __bang_le

```
void __bang_le(bfloat16_t *dst,
               bfloat16_t *src0,
               bfloat16_t *src1,
               int elem_count)
```

```
void __bang_le(half *dst,
               half *src0,
               half *src1,
               int elem_count)
```

```
void __bang_le(float *dst,
               float *src0,
               float *src1,
               int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is less than or equal to that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is less than or equal to the element of `<src1>`, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first vector.
- [in] `src1`: The address of the second vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__bang_eq](#) for more details.

3.15.40 __bang_le_bitindex

```
void __bang_le_bitindex(bfloat16_t *dst,
                       bfloat16_t *src0,
                       bfloat16_t *src1,
                       int elem_count)
```

```
void __bang_le_bitindex(half *dst,
                       half *src0,
                       half *src1,
                       int elem_count)
```

```
void __bang_le_bitindex(float *dst,
                       float *src0,
                       float *src1,
                       int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely, if the element of `<src0>` is less than or equal to that of `<src1>`, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information. The comparison process is illustrated in the figure below, the comparison result between the first element `<element0>` of `<src0>` and the first element `<element0>` of `<src1>` will be saved in `<bit0>` of `<dst>`. The other elements do the same comparison in turn.

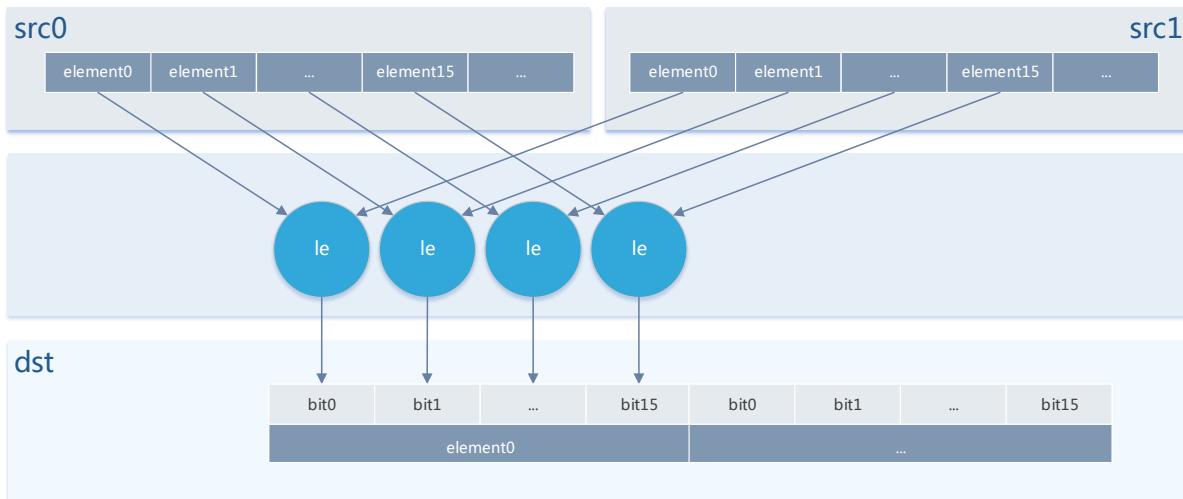


Fig. 3.9: The Calculation Process of Half Type `__bang_le_bitindex`

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src0>, <src1>` and `<dst>` must point to `__nram__` address space;
- `<elem_count>` must be divisible by 512 on `(m)tp_2xx`;
- `<elem_count>` must be divisible by 8 on `(m)tp_3xx` or higher;
- The address of `<src0>, <src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>, <src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_eq_bitindex` for more details.

3.15.41 __bang_le_scalar

```
void __bang_le_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_le_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_le_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are less than or equal to `<value>` and saves the result in `<dst>`. If the element of `<src>` is less than or equal to `<value>`, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_eq_scalar](#) for more details.

3.15.42 __bang_leu

```
void __bang_leu(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_leu(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_leu(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is unordered or less than or equal to that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is unordered or less than or equal to the element of `<src1>`, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

Cambricon@155chb

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first vector.
- [in] `src1`: The address of the second vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ](#) for more details.

3.15.43 __bang_leu_bitindex

```
void __bang_leu_bitindex(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_leu_bitindex(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_leu_bitindex(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely, if the element of `<src0>` is unordered or less than or equal to that of `<src1>`, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero and divisible by 8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ_bitindex](#) for more details.

3.15.44 __bang_leu_scalar

```
void __bang_leu_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_leu_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_leu_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are unordered or less than or equal to `<value>` and saves the result in `<dst>`. If the element of `<src>` is unordered or less than or equal to `<value>`, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

Cambricon@155chb

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of `__bang_equ_scalar` for more details.

3.15.45 __bang_lt

```
void __bang_lt(int *dst,
               int *src0,
               int *src1,
               int elem_count)
```

```
void __bang_lt(short *dst,
               short *src0,
               short *src1,
               int elem_count)
```

```
void __bang_lt(char *dst,
               char *src0,
               char *src1,
               int elem_count)
```

```
void __bang_lt(unsigned int *dst,
               unsigned int *src0,
               unsigned int *src1,
               int elem_count)
```

```
void __bang_lt(unsigned short *dst,
               unsigned short *src0,
               unsigned short *src1,
               int elem_count)
```

```
void __bang_lt(unsigned char *dst,
               unsigned char *src0,
               unsigned char *src1,
               int elem_count)
```

```
void __bang_lt(bfloat16_t *dst,
               bfloat16_t *src0,
               bfloat16_t *src1,
               int elem_count)
```

```
void __bang_lt(half *dst,
               half *src0,
               half *src1,
               int elem_count)
```

```
void __bang_lt(float *dst,
               float *src0,
               float *src1,
               int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is less than that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is less than the element of `<src1>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first vector.
- [in] `src1`: The address of the second vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_eq](#) for more details.

3.15.46 __bang_lt_bitindex

```
void __bang_lt_bitindex(bfloat16_t *dst,
                       bfloat16_t *src0,
                       bfloat16_t *src1,
                       int elem_count)
```

```
void __bang_lt_bitindex(half *dst,
                       half *src0,
                       half *src1,
                       int elem_count)
```

```
void __bang_lt_bitindex(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely, if the element of `<src0>` is less than that of `<src1>`, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information. The comparison process is illustrated in the figure below, the comparison result between the first element `<element0>` of `<src0>` and the first element `<element0>` of `<src1>` will be saved in `<bit0>` of `<dst>`. The other elements do the same comparison in turn.

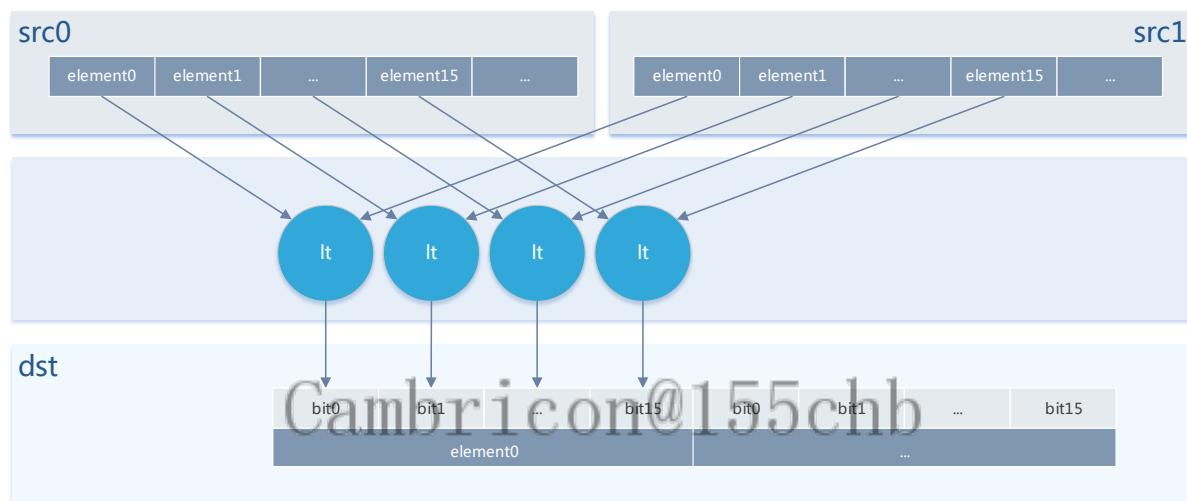


Fig. 3.10: The Calculation Process of Half Type `__bang_lt_bitindex`

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src0>`, `<src1>` and `<dst>` must point to `_nram_` address space;
- `<elem_count>` must be divisible by 512 on `(m)tp_2xx`;
- `<elem_count>` must be divisible by 8 on `(m)tp_3xx` or higher;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_eq_bitindex](#) for more details.

3.15.47 __bang_lt_scalar

```
void __bang_lt_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_lt_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_lt_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_lt_scalar(unsigned int *dst,
                      unsigned int *src,
                      unsigned int value,
                      int elem_count)
```

```
void __bang_lt_scalar(unsigned short *dst,
                      unsigned short *src,
                      unsigned short value,
                      int elem_count)
```

```
void __bang_lt_scalar(unsigned char *dst,
                      unsigned char *src,
                      unsigned char value,
                      int elem_count)
```

```
void __bang_lt_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_lt_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_lt_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are less than `<value>` and saves the result in `<dst>`. If the element of `<src>` is less than `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

Cambricon@155chb

- `<src>` and `<dst>` must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_eq_scalar](#) for more details.

3.15.48 __bang_ltu

```
void __bang_ltu(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_ltu(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_ltu(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely to determine whether the element in `<src0>` is unordered or less than that in `<src1>` and saves the result in `<dst>`. If the element of `<src0>` is unordered or less than the element of `<src1>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

`Cambricon@155chb`

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first vector.
- [in] `src1`: The address of the second vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of `__bang_equ` for more details.

3.15.49 __bang_ltu_bitindex

```
void __bang_ltu_bitindex(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_ltu_bitindex(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_ltu_bitindex(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

This function compares with `<elem_count>` elements in `<src0>` and `<src1>` element-wisely, if the element of `<src0>` is unordered or less than that of `<src1>`, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero and divisible by 8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ_bitindex](#) for more details.

3.15.50 __bang_ltu_scalar

```
void __bang_ltu_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_ltu_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_ltu_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are unordered or less than `<value>` and saves the result in `<dst>`. If the element of `<src>` is unordered or less than `<value>`, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

- See the example of [__bang_equ_scalar](#) for more details.

3.15.51 __bang_max

```
void __bang_max(half *dst,
                 half *src,
                 int elem_count)

void __bang_max(bfloat16_t *dst,
                 bfloat16_t *src,
                 int elem_count)

void __bang_max(float *dst,
                 float *src,
                 int elem_count)
```

Finds the maximum in a given vector. The result is composed of two parts. The first part is the maximum value of corresponding data type, the second part is the index of the maximum value in <src> vector, whose data type is `unsigned int`. The maximum value and index are stored continuously in <dst>. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `_nram_` address space;
- The `_nram_` address space to which `<dst>` points must be at least 128 bytes on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` cannot be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 128
```

```
--mlu_entry__ void kernel(float *c, float *a) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_max(c_tmp, a_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.15.52 __bang_maxeq_scalar

```
void __bang_maxeq_scalar(int *dst,
                         int *src,
                         int value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(short *dst,
                         short *src,
                         short value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(char *dst,
                         char *src,
                         char value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(unsigned int *dst,
                         unsigned int *src,
                         unsigned int value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(unsigned short *dst,
                         unsigned short *src,
                         unsigned short value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(unsigned char *dst,
                         unsigned char *src,
                         unsigned char value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(half *dst,
                         half *src,
                         half value,
                         int elem_count)
```

```
void __bang_maxeq_scalar(float *dst,
                         float *src,
                         float value,
                         int elem_count)
```

This function finds the maximum between `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_eq_scalar](#) for more details.

3.15.53 __bang_maxequal

```
void __bang_maxequal(int *dst,
                     int *src0,
                     int *src1,
                     int elem_count)
```

```
void __bang_maxequal(short *dst,
                     short *src0,
                     short *src1,
                     int elem_count)
```

```
void __bang_maxequal(char *dst,
                     char *src0,
                     char *src1,
                     int elem_count)
```

```
void __bang_maxequal(unsigned int *dst,
                     unsigned int *src0,
                     unsigned int *src1,
                     int elem_count)
```

```
void __bang_maxequal(unsigned short *dst,
                     unsigned short *src0,
                     unsigned short *src1,
                     int elem_count)
```

```
void __bang_maxequal(unsigned char *dst,
                     unsigned char *src0,
                     unsigned char *src1,
                     int elem_count)
```

```
void __bang_maxequal(half *dst,
                     half *src0,
                     half *src1,
                     int elem_count)
```

```
void __bang_maxequal(bfloat16_t *dst,
                     bfloat16_t *src0,
                     bfloat16_t *src1,
                     int elem_count)
```

```
void __bang_maxequal(float *dst,
                     float *src0,
                     float *src1,
                     int elem_count)
```

Finds maximum value of each two corresponding elements in the two vectors. If one of the elements being compared is NaN, then the value in <src0> is returned.

Parameters

- [out] dst: The address of destination vector
- [in] src0: The address of first source vector
- [in] src1: The address of second source vector
- [in] elem_count: The number of elements in source vector

Return

- void.

Remark

- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;

- <dst> can be overlapped with <src0>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.15.54 __bang_maximum

```
void __bang_maximum(float *dst,
                     float *src0,
                     float *src1,
                     int elem_count)

void __bang_maximum(half *dst,
                     half *src0,
                     half *src1,
                     int elem_count)

void __bang_maximum(bfloat16_t *dst,
                     bfloat16_t *src0,
                     bfloat16_t *src1,
                     int elem_count)
```

Compares two vectors and returns a new vector containing the element-wise maxima. If both of the elements being compared are NaN, then NaN is returned.

Parameters

- [out] dst: The address of destination vector
- [in] src0: The address of first source vector
- [in] src1: The address of second source vector
- [in] elem_count: The number of elements in source vector

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src0> or <src1>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- None.

3.15.55 __bang_min

```
void __bang_min(half *dst,
                 half *src,
                 int elem_count)

void __bang_min(bfloat16_t *dst,
                 bfloat16_t *src,
                 int elem_count)

void __bang_min(float *dst,
                 float *src,
                 int elem_count)
```

Finds the minimum value in `<src>` and stores the result in `<dst>`. The result consists of two parts, the first part is the minimum value of corresponding type, and the second part is the index of the minimum value, whose data type is `unsigned int`. The minimum value and index are stored continuously. See the table [Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- The `__nram__` address space to which `<dst>` points must be at least 128 bytes on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` cannot be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.15.56 __bang_mineq_scalar

```
void __bang_mineq_scalar(int *dst,
                          int *src,
                          int value,
                          int elem_count)
```

```
void __bang_mineq_scalar(short *dst,
                         short *src,
                         short value,
                         int elem_count)
```

```
void __bang_mineq_scalar(char *dst,
                         char *src,
                         char value,
                         int elem_count)
```

```
void __bang_mineq_scalar(unsigned int *dst,
                         unsigned int *src,
                         unsigned int value,
                         int elem_count)
```

```
void __bang_mineq_scalar(unsigned short *dst,
                         unsigned short *src,
                         unsigned short value,
                         int elem_count)
```

```
void __bang_mineq_scalar(unsigned char *dst,
                         unsigned char *src,
                         unsigned char value,
                         int elem_count)
```

```
void __bang_mineq_scalar(half *dst,
                         half *src,
                         half value,
                         int elem_count)
```

```
void __bang_mineq_scalar(float *dst,
                         float *src,
                         float value,
                         int elem_count)
```

This function finds the minimum between <elem_count> elements of <src> and <value> and

saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_eq_scalar](#) for more details.

3.15.57 __bang_minequal

```
void __bang_minequal(int *dst,
                      int *src0,
                      int *src1,
                      int elem_count)
```

```
void __bang_minequal(short *dst,
                      short *src0,
                      short *src1,
                      int elem_count)
```

```
void __bang_minequal(char *dst,
                      char *src0,
                      char *src1,
                      int elem_count)
```

```
void __bang_minequal(unsigned int *dst,
                      unsigned int *src0,
                      unsigned int *src1,
                      int elem_count)
```

```
void __bang_minequal(unsigned short *dst,
                      unsigned short *src0,
                      unsigned short *src1,
                      int elem_count)
```

```
void __bang_minequal(unsigned char *dst,
                      unsigned char *src0,
                      unsigned char *src1,
                      int elem_count)
```

```
void __bang_minequal(half *dst,
                      half *src0,
                      half *src1,
                      int elem_count)
```

```
void __bang_minequal(bfloat16_t *dst,
                      bfloat16_t *src0,
                      bfloat16_t *src1,
                      int elem_count)
```

```
void __bang_minequal(float *dst,
                      float *src0,
                      float *src1,
                      int elem_count)
```

Finds minimum value of each two corresponding elements in the two vectors. If one of the elements being compared is NAN, then the value in <src0> is returned.

Parameters

- [out] dst: The address of destination vector
- [in] src0: The address of first source vector
- [in] src1: The address of second source vector
- [in] elem_count: The number of elements in source vector

Return

- void.

Remark

- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src0>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;

- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(half* c, half* a, half* b) {
    __nram__ half a_tmp[DATA_SIZE];
    __nram__ half c_tmp[DATA_SIZE];
    __nram__ half b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __bang_minequal(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.15.58 __bang_minimum

```
void __bang_minimum(float *dst,
                    float *src0,
                    float *src1,
                    int elem_count)
```

```
void __bang_minimum(half *dst,
                    half *src0,
                    half *src1,
                    int elem_count)
```

```
void __bang_minimum(bfloat16_t *dst,
                    bfloat16_t *src0,
                    bfloat16_t *src1,
                    int elem_count)
```

Compares two vectors and returns a new vector containing the element-wise minima. If both of the elements being compared are NaN, then NaN is returned.

Parameters

- [out] dst: The address of destination vector
- [in] src0: The address of first source vector
- [in] src1: The address of second source vector
- [in] elem_count: The number of elements in source vector

Return

- void.

Remark

- <elem_count> must be greater than zero;

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src0> or <src1>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

- None.

3.15.59 __bang_nan_maximum

```
void __bang_nan_maximum(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_nan_maximum(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_nan_maximum(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

Cambricon@155chb

Compares two vectors and returns a new vector containing the element-wise maxima. If one of the elements being compared is a NaN, then that element is returned.

Parameters

- [out] dst: The address of destination vector
- [in] src0: The address of first source vector
- [in] src1: The address of second source vector
- [in] elem_count: The number of elements in source vector

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src0> or <src1>.
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.15.60 __bang_nan_minimum

```
void __bang_nan_minimum(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_nan_minimum(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_nan_minimum(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

Compares two vectors and returns a new vector containing the element-wise minima. If one of the elements being compared is a NaN, then that element is returned.

Parameters

- [out] dst: The address of destination vector
- [in] src0: The address of first source vector
- [in] src1: The address of second source vector
- [in] elem_count: The number of elements in source vector

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src0> or <src1>.
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.15.61 __bang_ne

```
void __bang_ne(int *dst,
               int *src0,
               int *src1,
               int elem_count)
```

```
void __bang_ne(short *dst,
               short *src0,
               short *src1,
               int elem_count)
```

```
void __bang_ne(char *dst,
               char *src0,
               char *src1,
               int elem_count)
```

```
void __bang_ne(half *dst,
               half *src0,
               half *src1,
               int elem_count)
```

```
void __bang_ne(bfloat16_t *dst,
               bfloat16_t *src0,
               bfloat16_t *src1,
               int elem_count)
```

```
void __bang_ne(float *dst,
               float *src0,
               float *src1,
               int elem_count)
```

This function performs inequality comparison with `<src0>` and `<src1>` element-wisely and saves the result in `<dst>`. If the element of `<src0>` and the element of `<src1>` are not equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `<src0>, <src1>` and `<dst>` must point to `_nram_` address space;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>, <src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>, <src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_eq](#) for more details.

3.15.62 __bang_ne_bitindex

```
void __bang_ne_bitindex(bfloat16_t *dst,
                       bfloat16_t *src0,
                       bfloat16_t *src1,
                       int elem_count)
```

```
void __bang_ne_bitindex(half *dst,
                       half *src0,
                       half *src1,
                       int elem_count)
```

```
void __bang_ne_bitindex(float *dst,
                       float *src0,
                       float *src1,
                       int elem_count)
```

This function performs inequality comparison with `<src0>` and `<src1>` element-wisely. If the two elements are not equal, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information. The comparison process is illustrated in the figure below, the comparison result between the first element `<element0>` of `<src0>` and the first element `<element0>` of `<src1>` will be saved in `<bit0>` of `<dst>`. The other elements do the same comparison in turn.

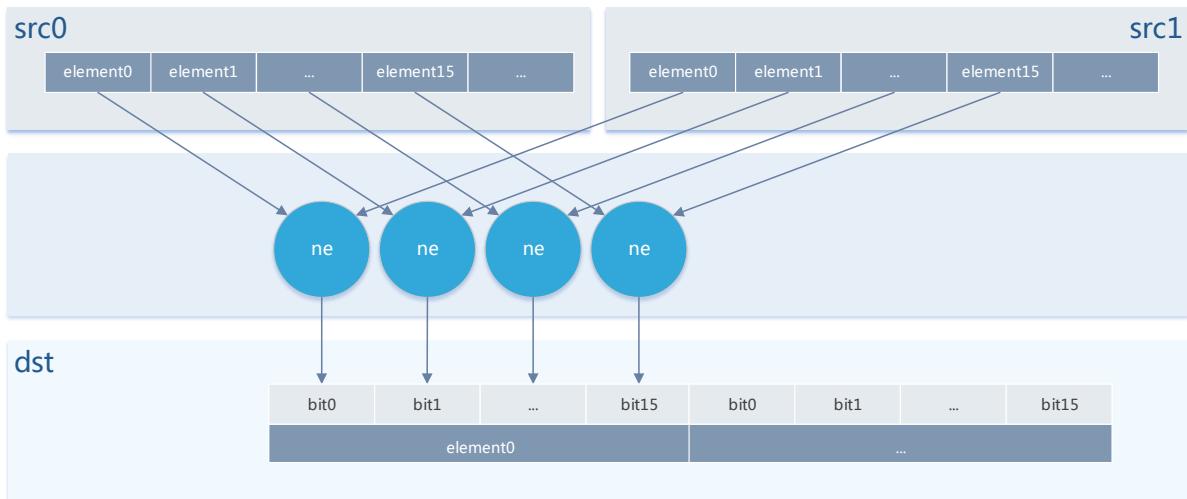


Fig. 3.11: The Calculation Process of Half Type `__bang_ne_bitindex`

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be divisible by 512 on `(m)tp_2xx`;
- `<elem_count>` must be divisible by 8 on `(m)tp_3xx` or higher;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_eq_bitindex` for more details.

3.15.63 __bang_ne_scalar

```
void __bang_ne_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)

void __bang_ne_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)

void __bang_ne_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)

void __bang_ne_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_ne_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_ne_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function performs inequality comparison with `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. The type of result is same as the type of `<src>`. If the element of `<src>` and `<value>` are not equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;

- `bfloat16_t` is supported on (m)tp_5xx or higher;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of `__bang_eq_scalar` for more details.

3.15.64 `__bang_neu`

```
void __bang_neu(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_neu(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_neu(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function performs unoredred or inequality comparison with `<src0>` and `<src1>` element-wisely and saves the result in `<dst>`. If the element of `<src0>` and the element of `<src1>` are unordered or not equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

- See the example of [__bang_equ](#) for more details.

3.15.65 __bang_neu_bitindex

```
void __bang_neu_bitindex(bfloat16_t *dst,
                        bfloat16_t *src0,
                        bfloat16_t *src1,
                        int elem_count)
```

```
void __bang_neu_bitindex(half *dst,
                        half *src0,
                        half *src1,
                        int elem_count)
```

```
void __bang_neu_bitindex(float *dst,
                        float *src0,
                        float *src1,
                        int elem_count)
```

This function performs unordered or inequality comparison with `<src0>` and `<src1>` element-wisely. If the two elements are unordered or not equal, the result is 1. Otherwise, the result is 0. The result will be sequentially saved in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count>` must be greater than zero and divisible by 8.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

- See the example of [__bang_equ_bitindex](#) for more details.

3.15.66 __bang_neu_scalar

```
void __bang_neu_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_neu_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_neu_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function performs unordered or inequality comparison with `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. The type of result is same as the type of `<src>`. If the element of `<src>` and `<value>` are unordered or not equal, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

- See the example of [__bang_equ_scalar](#) for more details.

3.16 Vector Fusion Functions

3.16.1 __bang fabsmax

```
void __bang fabsmax(half *dst,
                     half *src,
                     int elem_count)
```

```
void __bang fabsmax(bfloat16_t *dst,
                     bfloat16_t *src,
                     int elem_count)
```

```
void __bang fabsmax(float *dst,
                     float *src,
                     int elem_count)
```

This function performs calculation of the absolute value of `<src>` element-wisely, then finds the maximum absolute value and its position index, and saves the maximum absolute value and its index in `<dst>`. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<dst>` and `<src>` must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- For `half` type, the first 2 bytes of `<dst>` are used to record the maximum absolute value (`half` type), and the next 8 bytes are occupied as an index to locate the position of the value. Only the low 22 bits of the 8 bytes index are valid as an integer index, and the high 42 bits are all written to zero;
- For `float` type, the first 4 bytes of `<dst>` are used to record the maximum absolute value (`float` type), and the next 8 bytes are occupied as an index to locate the position of the value. Only the low 26 bits of the 8 bytes index are valid as an integer index, and the high 38 bits are all written to zero;
- `<dst>` will occupy 10 bytes for `half` type and 12 bytes for `float` type;
- When `<src>` vector has multiple same maximum absolute values, only the first one and its index will be stored;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define RES_ELEM_NUM 16
#define ELEM_NUM 128

__mlu_entry__ void kernel fabs(float* dst, float *src) {
    __nram__ float dst_nram[RES_ELEM_NUM];
    __nram__ float src_nram[ELEM_NUM];
    __memcpy(src_nram, src, ELEM_NUM * sizeof(float), GDRAM2NRAM);
    __bang_fabsmax(dst_nram, src_nram, ELEM_NUM);
    __memcpy(dst, dst_nram, RES_ELEM_NUM * sizeof(float), NRAM2GDRAM);
}
```

3.16.2 __bang_fabsmin

Cambricon@155chb

```
void __bang_fabsmin(half *dst,
                     half *src,
                     int elem_count)
```

```
void __bang_fabsmin(bfloat16_t *dst,
                     bfloat16_t *src,
                     int elem_count)
```

```
void __bang_fabsmin(float *dst,
                     float *src,
                     int elem_count)
```

This function performs calculation of the absolute value of `<src>` element-wisely, then finds the minimum absolute value and its position index, and saves the minimum absolute value and its index in `<dst>`. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- <dst> and <src> must point to `__nram__` address space;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- For `half` and `bfloat16_t` type, the first 2 bytes of <dst> are used to record the minimum absolute value (`half` type), and the next 8 bytes are occupied as an index to locate the position of the value. Only the low 22 bits of the 8 bytes index are valid as an integer index, and the high 42 bits are all written to zero;
- For `float` type, the first 4 bytes of <dst> are used to record the minimum absolute value (`float` type), and the next 8 bytes are occupied as an index to locate the position of the value. Only the low 26 bits of the 8 bytes index are valid as an integer index, and the high 38 bits are all written to zero;
- <dst> will occupy 10 bytes for `half` type and 12 bytes for `float` type;
- When <src> vector has multiple same minimum absolute values, only the first one and its index will be stored;
- <dst> can be overlapped with <src>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of `__bang_tabsmax` for more details.

3.16.3 `__bang_fcmpfilter`

```
void __bang_fcmpfilter(CompareMode mode,
                      half *dst,
                      half *src0,
                      half *src1,
                      int elem_count)
```

```
void __bang_fcmpfilter(CompareMode mode,
                      half *dst,
                      half *src0,
                      half src1,
                      int elem_count)
```

```
void __bang_fcmpfilter(CompareMode mode,
                      float *dst,
                      float *src0,
                      float *src1,
                      int elem_count)
```

```
void __bang_fcmpfilter(CompareMode mode,
                      bfloat16_t *dst,
                      bfloat16_t *src0,
                      bfloat16_t src1,
                      int elem_count)
```

```
void __bang_fcmpfilter(CompareMode mode,
                      bfloat16_t *dst,
                      bfloat16_t *src0,
                      bfloat16_t *src1,
                      int elem_count)
```

```
void __bang_fcmpfilter(CompareMode mode,
                      float *dst,
                      float *src0,
                      float src1,
                      int elem_count)
```

This function performs comparison with `<src0>` and `<src1>` element-wisely according to `<mode>` and saves the result in `<dst>`. `<mode>` indicates the type of comparison. `<mode>` is assigned to an enumerated type called `CompareMode` that contains six enumerators listed in the table below.

Cambricon@155chb

Table 3.39: Semantics of CompareMode

CompareMode Type	Semantic
CMP_EQ	=
CMP_NE	≠
CMP_LT	<
CMP_LE	≤
CMP_GT	>
CMP_GE	≥

See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [in] mode: The comparison mode.
- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector or the second source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0> and <dst> must point to __nram__ address space. <src1> must also point to __nram__ address space if it is a vector;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <src1> can be either vectors or scalars. When <src1> is a vector, the length of <src1> must be the same as that of <src0>;
- <dst> can be overlapped with <src0>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define ELEM_NUM 128
#define RES_ELEM_NUM (ELEM_NUM + 4)

__mlu_entry__ void kernel fabs(CompareMode mode, float* dst,
                               float* src0, float* src1) {
    __nram__ float dst_nram[RES_ELEM_NUM];
    __nram__ float src0_nram[ELEM_NUM];
    __nram__ float src1_nram[ELEM_NUM];
    __memcpy(src0_nram, src0, ELEM_NUM * sizeof(float), GDRAM2NRAM);
    __memcpy(src1_nram, src1, ELEM_NUM * sizeof(float), GDRAM2NRAM);
    __bang_fcmpfilter(mode, dst_nram, src0_nram, src1_nram, ELEM_NUM);
    __memcpy(dst, dst_nram, RES_ELEM_NUM * sizeof(float), NRAM2GDRAM);
}
```

3.16.4 __bang_fusion

```
void __bang_fusion(mluFusionOpCode op_code,
                   half *dst,
                   half *src0,
                   half src1,
                   half src2,
                   int src_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   float *dst,
                   float *src0,
                   float *src1,
                   float src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   float *dst,
                   float *src0,
                   float src1,
                   float *src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   float *dst,
                   float *src0,
                   float src1,
                   float src2,
                   int src_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   half *dst,
                   half *src0,
                   half *src1,
                   half *src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   half *dst,
                   half *src0,
                   half *src1,
                   half src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   half *dst,
                   half *src0,
                   half src1,
                   half *src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   bf16_t *dst,
                   bf16_t *src0,
                   bf16_t *src1,
                   bf16_t *src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   bf16_t *dst,
                   bf16_t *src0,
                   bf16_t *src1,
                   bf16_t src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   bf16_t *dst,
                   bf16_t *src0,
                   bf16_t src1,
                   bf16_t *src2,
                   int src_elem_count,
                   int seg_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   bf16_t *dst,
                   bf16_t *src0,
                   bf16_t src1,
                   bf16_t src2,
                   int src_elem_count)
```

```
void __bang_fusion(mluFusionOpCode op_code,
                   float *dst,
                   float *src0,
                   float *src1,
                   float *src2,
                   int src_elem_count,
                   int seg_elem_count)
```

This function performs fused arithmetic calculation on vectors $\langle \text{src0} \rangle$, $\langle \text{src1} \rangle$ and $\langle \text{src2} \rangle$ element-wisely according to $\langle \text{op_code} \rangle$ and saves the result in $\langle \text{dst} \rangle$. $\langle \text{op_code} \rangle$ indicates the type of operator. $\langle \text{op_code} \rangle$ is assigned to an enumerated type called `mluFusionOpCode` that contains eight enumerators listed in the table below.

Table 3.40: Semantics of `mluFusionOpCode`

<code>mluFusionOpCode</code> Type	Semantic
FUSION_FMA	$\langle \text{dst} \rangle = \langle \text{src0} \rangle \times \langle \text{src1} \rangle + \langle \text{src2} \rangle$
FUSION_FMS	$\langle \text{dst} \rangle = \langle \text{src0} \rangle \times \langle \text{src1} \rangle - \langle \text{src2} \rangle$
FUSION_FAM	$\langle \text{dst} \rangle = (\langle \text{src0} \rangle + \langle \text{src1} \rangle) \times \langle \text{src2} \rangle$
FUSION_FSM	$\langle \text{dst} \rangle = (\langle \text{src0} \rangle - \langle \text{src1} \rangle) \times \langle \text{src2} \rangle$
FUSION_FAA	$\langle \text{dst} \rangle = \langle \text{src0} \rangle + \langle \text{src1} \rangle + \langle \text{src2} \rangle$
FUSION_FAS	$\langle \text{dst} \rangle = \langle \text{src0} \rangle + \langle \text{src1} \rangle - \langle \text{src2} \rangle$
FUSION_FSS	$\langle \text{dst} \rangle = \langle \text{src0} \rangle - \langle \text{src1} \rangle - \langle \text{src2} \rangle$
FUSION_FSA	$\langle \text{dst} \rangle = \langle \text{src0} \rangle - \langle \text{src1} \rangle + \langle \text{src2} \rangle$

Note:

For the last two characters of enumerators, ‘M’ means multiplication, ‘A’ means addition and ‘S’ means subtraction.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [in] `op_code`: The type of operator.
- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The second source scalar or the address of the second source vector.
- [in] `src2`: The third source scalar or the address of the third source vector.
- [in] `src_elem_count`: The number of elements in $\langle \text{dst} \rangle$ and $\langle \text{src0} \rangle$.
- [in] `seg_elem_count`: The number of elements in $\langle \text{src1} \rangle$ and $\langle \text{src2} \rangle$.

Return

- `void`.

Remark

- <dst> can be overlapped with <src0>;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <src0> and <dst> must point to __nram__ address space;
- <src1> or <src2> must also point to __nram__ address space if it is a vector;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src_elem_count> and <seg_elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define BUF_SIZE 128

__mlu_entry__ void kernel(float *mlu_dst,
                          float *mlu_src0,
                          float *mlu_src1,
                          float *mlu_src2) {
    __nram__ float nram_output[BUF_SIZE];
    __nram__ float nram_input0[BUF_SIZE];
    __nram__ float nram_input1[BUF_SIZE];
    __nram__ float nram_input2[BUF_SIZE];
    __memcpy(nram_input0, mlu_src0, BUF_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(nram_input1, mlu_src1, BUF_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(nram_input2, mlu_src2, BUF_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_fusion(FUSION_FMA, nram_output, nram_input0,
                  nram_input1, nram_input2, BUF_SIZE, BUF_SIZE);
    __memcpy(mlu_dst, nram_output, BUF_SIZE, NRAM2GDRAM);
}
```

3.17 Vector Logic Functions

3.17.1 __bang_and

```
void __bang_and(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```

void __bang_and(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)

void __bang_and(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)

void __bang_and(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)

void __bang_and(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)

void __bang_and(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)

```

Performs logical AND operation between elements in two vectors. If both the elements of <src0> and <src1> are non-zero, the result is 1. Otherwise, the result is 0. The type of result is the same as that of <src0> and <src1>. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src0>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(half* c, half* a, half* b) {
    __nram__ half a_tmp[DATA_SIZE];
    __nram__ half c_tmp[DATA_SIZE];
    __nram__ half b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __bang_and(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.17.2 __bang_and_scalar

```
void __bang_and_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_and_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_and_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_and_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_and_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_and_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function performs logical AND operation between `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. If both the element of `<src>` and `<value>` are non-zero, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

Cambricon@155chb

- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_and_scalar(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
```

```
}
```

3.17.3 __bang_cycle_and

```
void __bang_cycle_and(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_and(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_and(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_and(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_and(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_and(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts. Each element in each part AND the corresponding element in `<seg>`. The result is assigned to `<dst>`. If both the elements of `<src0>` and `<src1>` are non-zero, the result is 1. Otherwise, the result is 0. The type of result is the same as that of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <src_elem_count> * sizeof(type) and <seg_elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- <src>, <seg> and <dst> must point to __nram__ address space;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define N1 128
#define N2 32640

__mlu_entry__ void kernel(float *c, float *a, float *b) {
    __nram__ float a_tmp[N1];
    __nram__ float b_tmp[N2];
    __nram__ float c_tmp[N2];
    __memcpy(a_tmp, a, N1 * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, N2 * sizeof(float), GDRAM2NRAM);
    __bang_cycle_and(c_tmp, b_tmp, a_tmp, N2, N1);
    __memcpy(c, c_tmp, N2 * sizeof(float), NRAM2GDRAM);
}
```

3.17.4 __bang_cycle_or

```
void __bang_cycle_or(half *dst,
                     half *src,
                     half *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_or(int *dst,
                     int *src,
                     int *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_or(short *dst,
                     short *src,
                     short *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_or(char *dst,
                     char *src,
                     char *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_or(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

```
void __bang_cycle_or(float *dst,
                     float *src,
                     float *seg,
                     int src_elem_count,
                     int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts. Each element in each part OR the corresponding element in `<seg>`. The result is assigned to `<dst>`. If both the elements of `<src0>` and `<src1>` are zero, the result is 0. Otherwise, the result is 1. The type of result is the same as that of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.

- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in <src> vector.
- [in] `seg_elem_count`: The number of elements in <seg> vector.

Return

- `void`.

Remark

- `int`, `short` and `char` are supported on (m)tp_3xx or higher;
- `bfloat16_t` is supported on (m)tp_5xx or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on (m)tp_2xx;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on (m)tp_2xx;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

Cambricon@155chb

3.17.5 __bang_cycle_xor

```
void __bang_cycle_xor(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_xor(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_xor(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_xor(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_xor(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_xor(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count>` / `<seg_elem_count>` parts. Each element in each part XOR the corresponding element in `<seg>`. The result is assigned to `<dst>`. If both the elements of `<src0>` and `<src1>` are zero or non-zero, the result is 0. Otherwise, the result is 1. The type of result is the same as that of `<src0>` and `<src1>`. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;

- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.17.6 __bang_not

```
void __bang_not(half *dst,
                 half *src,
                 int elem_count)

void __bang_not(int *dst,
                 int *src,
                 int elem_count)

void __bang_not(short *dst,
                 short *src,
                 int elem_count)

void __bang_not(char *dst,
                 char *src,
                 int elem_count)

void __bang_not(bfloat16_t *dst,
                 bfloat16_t *src,
                 int elem_count)

void __bang_not(float *dst,
                 float *src,
                 int elem_count)
```

Cambricon@155chb

Applies element-wisely NOT operation on a vector. For each element in `<src>`, if it equals to zero, then stores 1 at the corresponding position in `<dst>`; otherwise, stores 0 at the position in `<dst>`. The type of result is the same as that of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Unary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.17.7 __bang_or

```
void __bang_or(half *dst,
               half *src0,
               half *src1,
               int elem_count)
```

```
void __bang_or(int *dst,
               int *src0,
               int *src1,
               int elem_count)
```

```
void __bang_or(short *dst,
               short *src0,
               short *src1,
               int elem_count)
```

```
void __bang_or(char *dst,
               char *src0,
               char *src1,
               int elem_count)
```

```
void __bang_or(bfloat16_t *dst,
               bfloat16_t *src0,
               bfloat16_t *src1,
               int elem_count)
```

```
void __bang_or(float *dst,
               float *src0,
               float *src1,
               int elem_count)
```

Performs logical OR operation between elements in two vectors. If both the elements of <src0> and <src1> are zero, the result is 0. Otherwise, the result is 1. The type of result is the same as that of <src0> and <src1>. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <elem_count> must be greater than zero;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: --BANG_ARCH-- >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.17.8 __bang_or_scalar

```
void __bang_or_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_or_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_or_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_or_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_or_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_or_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function performs logical OR operation between `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. If either the element of `<src>` or `<value>` is non-zero, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_and_scalar](#) for more details.

3.17.9 __bang_xor

```
void __bang_xor(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)

void __bang_xor(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)

void __bang_xor(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)

void __bang_xor(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)

void __bang_xor(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)

void __bang_xor(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

Performs logical XOR operation between elements in two vectors. If both the elements of <src0> and <src1> are zero or non-zero, the result is 0. Otherwise, the result is 1. The type of result is the same as that of <src0> and <src1>. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;

- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<elem_count>` must be greater than zero;
- `<src0>, <src1>` and `<dst>` must point to `__nram__` address space;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src0>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(half* c, half* a, half* b) {
    __nram__ half a_tmp[DATA_SIZE];
    __nram__ half c_tmp[DATA_SIZE];
    __nram__ half b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __bang_xor(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.17.10 `__bang_xor_scalar`

```
void __bang_xor_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_xor_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_xor_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_xor_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_xor_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_xor_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function performs logical OR operation between `<elem_count>` elements of `<src>` and `<value>` and saves the result in `<dst>`. If the element of `<src>` is zero and `<value>` is non-zero, or the element of `<src>` is non-zero and `<value>` is zero, the result is 1. Otherwise, the result is 0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src>` and `<dst>` must point to `_nram_` address space;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [`__bang_and_scalar`](#) for more details.

3.18 Vector Movement Functions

3.18.1 __bang_collect

```
void __bang_collect(half *dst,
                     half *src,
                     half *mask,
                     int elem_count)
```

```
void __bang_collect(bfloat16_t *dst,
                     bfloat16_t *src,
                     bfloat16_t *mask,
                     int elem_count)
```

```
void __bang_collect(float *dst,
                     float *src,
                     float *mask,
                     int elem_count)
```

Selects number in one vector according to the corresponding values in another vector. The elements in <src> will be selected and stored continuously if corresponding elements in <mask> are not equal to zero. The result is the selected elements. The selected elements will be stored continuously.

Cambricon@155chb

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] mask: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src>;
- The <src>, <mask> and <dst> must point to __nram__ address space;
- <elem_count> * sizeof(type) must be a multiple of 128 bytes on (m)tp_2xx;
- The address of <src>, <mask> and <dst> must be 64-byte aligned on (m)tp_2xx;
- On (m)tp_2xx, if the total number of bytes of the selected elements is not a multiple of 128, it will be aligned to 128 bytes, and the padding bytes will be set to zero;
- On (m)tp_2xx, the size of <dst> cannot be smaller than the size of <src> and if the selected data does not fill the entire <dst> space, the remaining data in <dst> may also be changed.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;

- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.18.2 __bang_collect_bitindex

```
void __bang_collect_bitindex(half *dst,
                             half *src,
                             void *bitmask,
                             int elem_count)

void __bang_collect_bitindex(bfloat16_t *dst,
                            bfloat16_t *src,
                            void *bitmask,
                            int elem_count)

void __bang_collect_bitindex(float *dst,
                            float *src,
                            void *bitmask,
                            int elem_count)
```

Selects the corresponding elements in `<src>` according to `<bitmask>`. The elements in `<src>` will be saved to `<dst>`, if corresponding bit in `<bitmask>` is 1. All selected elements will be stored continuously in `<dst>`. The selected elements will be stored continuously. The behavior is illustrated in the figure below, in this example, `<bitmask>` is `0x41100000`.

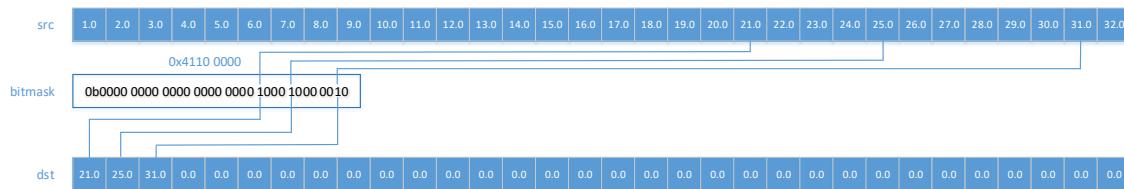


Fig. 3.12: The Calculation Process of float Type __bang_collect_bitindex

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `bitmask`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<dst>` can be overlapped with `<src>`;
- The `<src>`, `<bitmask>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>`, `<bitmask>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<elem_count>` must be divisible by 512 on `(m)tp_2xx`, and divisible by 8 on `mtp_372` and `tp_322`;
- On `(m)tp_2xx`, if the total number of bytes of the selected elements is not a multiple of 128, it will be aligned to 128 bytes, and the padding bytes will be set to zero;
- On `(m)tp_2xx`, the size of `<dst>` cannot be smaller than the size of `<src>` and if the selected data does not fill the entire `<dst>` space, the remaining data in `<dst>` may also be changed.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.18.3 `__bang_maskmove`

```
void __bang_maskmove(void *dst,
                     void *src,
                     void *mask,
                     int elem_count)
```

Selects bytes in `<src>`, whose length is `<elem_count>`, according to the bit value of the vector `<mask>`, and stores the result in `<dst>`. The bytes in `<src>` will be selected if corresponding bit in `<mask>` is 1, otherwise, keeps the corresponding byte in `<dst>` unchanged.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `mask`: The address of mask vector.
- [in] `elem_count`: The length of source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<elem_count>` must be divisible by 1024 on `(m)tp_2xx`;
- `<elem_count>` must be divisible by 8 on `(m)tp_3xx`;
- `<src>`, `<mask>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>`, `<mask>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.18.4 `__bang_mirror`

```
void __bang_mirror(int8_t *dst,
                   int8_t *src,
                   int height,
                   int width)
```

```
void __bang_mirror(char *dst,
                   char *src,
                   int height,
                   int width)
```

```
void __bang_mirror(unsigned char *dst,
                   unsigned char *src,
                   int height,
                   int width)
```

```
void __bang_mirror(unsigned short *dst,
                   unsigned short *src,
                   int height,
                   int width)
```

```
void __bang_mirror(short *dst,
                   short *src,
                   int height,
                   int width)
```

```
void __bang_mirror(half *dst,
                   half *src,
                   int height,
                   int width)
```

```
void __bang_mirror(unsigned int *dst,
                   unsigned int *src,
                   int height,
                   int width)
```

Flips <src>, a matrix whose size is <height> * <width>, in the left/right direction, and stores the result in <dst>.

Parameters

- [out] dst: The address of destination matrix.
- [in] src: The address of source matrix.
- [in] height: The width of <src>.
- [in] width: The height of <src>.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <height> * sizeof(type) must be divisible by 64 on (m)tp_2xx;
- <width> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> cannot be overlapped with <src>;
- <height> and <width> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define WIDTH 64
#define HEIGHT 32
#define LEN (WIDTH * HEIGHT)

__mlu_entry__ void kernel(short* dst, short* src) {
    __nram__ short ny[LEN];
    __nram__ short nx[LEN];
    __memcpy(nx, src, LEN * sizeof(short), GDRAM2NRAM);
    __bang_mirror(ny, nx, HEIGHT, WIDTH);
    __memcpy(dst, ny, sizeof(short) * LEN, NRAM2GDRAM);
}
```

3.18.5 __bang_pad

```
void __bang_pad(half *dst,  
                 half *src,  
                 int channel,  
                 int height,  
                 int width,  
                 int pad_top,  
                 int pad_bottom,  
                 int pad_left,  
                 int pad_right)
```

```
void __bang_pad(short *dst,  
                 short *src,  
                 int channel,  
                 int height,  
                 int width,  
                 int pad_top,  
                 int pad_bottom,  
                 int pad_left,  
                 int pad_right)
```

```
void __bang_pad(unsigned short *dst,  
                 unsigned short *src,  
                 int channel,  
                 int height,  
                 int width,  
                 int pad_top,  
                 int pad_bottom,  
                 int pad_left,  
                 int pad_right)
```

```
void __bang_pad(int8_t *dst,  
                 int8_t *src,  
                 int channel,  
                 int height,  
                 int width,  
                 int pad_top,  
                 int pad_bottom,  
                 int pad_left,  
                 int pad_right)
```

```
void __bang_pad(char *dst,
                 char *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right)
```

```
void __bang_pad(unsigned char *dst,
                 unsigned char *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right)
```

```
void __bang_pad(float *dst,
                 float *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right)
```

```
void __bang_pad(unsigned int *dst,
                 unsigned int *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right)
```

```
void __bang_pad(half *dst,
                 half *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right,
                 half pad_value)
```

```
void __bang_pad(short *dst,
                 short *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right,
                 short pad_value)
```

```
void __bang_pad(char *dst,
                 char *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right,
                 char pad_value)
```

```
void __bang_pad(float *dst,
                 float *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right,
                 float pad_value)
```

```
void __bang_pad(int *dst,
                 int *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right,
                 int pad_value)
```

```
void __bang_pad(bfloat16_t *dst,
                 bfloat16_t *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right,
                 bfloat16_t pad_value)
```

```
void __bang_pad(int *dst,
                 int *src,
                 int channel,
                 int height,
                 int width,
                 int pad_top,
                 int pad_bottom,
                 int pad_left,
                 int pad_right)
```

Applies padding operation on <src>.

Parameters

- [out] dst: The destination vector, whose data layout is HWC.
- [in] src: The source vector, whose data layout is HWC.
- [in] channel: Number of channels.
- [in] height: The height of <src>.
- [in] width: The width of <src>.
- [in] pad_top: Number of rows whose elements is all zero or <pad_value> on the top of pad.
- [in] pad_bottom: Number of rows whose elements is all zero or <pad_value> on the bottom of pad.
- [in] pad_left: Number of columns whose elements is all zero or <pad_value> on the left of pad.

- [in] `pad_right`: Number of columns whose elements is all zero or `<pad_value>` on the right of pad.
- [in] `pad_value`: The value of padding.

Return

- `void`.

Remark

- `<height>` and `<width>` must be greater than 0;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<dst>` cannot be overlapped with `<src>`;
- `<src>` and `<dst>` must point to `_nram_` address space;
- If `<height> == 1`, `<pad_left>` and `<pad_right>` must be equal to 0;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<channel> * <width> * sizeof(type)` must be 128-byte aligned on `(m)tp_2xx`;
- `<pad_left> * <channel> * sizeof(type)` must be 128-byte aligned on `(m)tp_2xx`;
- `<pad_right> * <channel> * sizeof(type)` must be 128-byte aligned on `(m)tp_2xx`;
- `<pad_value>` are only supported on `(m)tp_5xx` or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

Cambricon@155chb

```
#include<bang.h>

#define PAD_H 1
#define PAD_W 2
#define INPUT_H 4
#define INPUT_W 4
#define CHANNEL 64

__mlu_entry__ void kernel(half *out, half *in) {
    __nram__ half nx[CHANNEL * INPUT_H * INPUT_W];
    __nram__ half ny[CHANNEL * (INPUT_H + 2 * PAD_H) * (INPUT_W + 2 * PAD_W)];
    __memcpy(nx, in, CHANNEL * INPUT_H * INPUT_W * sizeof(half), GDRAM2NRAM);
    __bang_pad(ny, nx, CHANNEL, INPUT_H, INPUT_W, PAD_H, PAD_H, PAD_W, PAD_W);
    __memcpy(out, ny, CHANNEL * (INPUT_H + 2 * PAD_H) * (INPUT_W + 2 * PAD_W) * sizeof(half), NRAM2GDRAM);
}
```

3.18.6 __bang_rotate180

```
void __bang_rotate180(int8_t *dst,  
                      int8_t *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(char *dst,  
                      char *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(unsigned char *dst,  
                      unsigned char *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(half *dst,  
                      half *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(short *dst,  
                      short *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(unsigned short *dst,  
                      unsigned short *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(float *dst,  
                      float *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(int *dst,  
                      int *src,  
                      int height,  
                      int width)  
  
void __bang_rotate180(unsigned int *dst,  
                      unsigned int *src,  
                      int height,  
                      int width)
```

Rotates <src>, a matrix, whose size is <height> * <width>, by 180 degrees in clockwise direction, and stores the result in <dst>.

Parameters

- [out] dst: The address of destination matrix.
- [in] src: The address of source matrix.
- [in] height: The height of <src>.
- [in] width: The width of <src>.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <height> * sizeof(type) and <width> * sizeof(type) must be divisible by 128 bytes on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> cannot be overlapped with <src>;
- <height> and <width> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.7 __bang_rotate270

```
void __bang_rotate270(int8_t *dst,
                      int8_t *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(char *dst,
                      char *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(unsigned char *dst,
                      unsigned char *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(half *dst,
                      half *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(short *dst,
                      short *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(unsigned short *dst,
                      unsigned short *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(float *dst,
                      float *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(int *dst,
                      int *src,
                      int height,
                      int width)
```

```
void __bang_rotate270(unsigned int *dst,
                      unsigned int *src,
                      int height,
                      int width)
```

Rotates <src> , a matrix, whose size is <height> * <width> , by 270 degrees in clockwise direction, and stores the result in <dst> .

Parameters

- [out] dst: The address of destination matrix.
- [in] src: The address of source matrix.
- [in] height: The height of <src>.
- [in] width: The width of <src>.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <height> * sizeof(type) and <width> * sizeof(type) must be divisible by 128 bytes on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> cannot be overlapped with <src>;
- <height> and <width> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.18.8 `__bang_rotate90`

```
void __bang_rotate90(int8_t *dst,
                      int8_t *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(char *dst,
                      char *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(unsigned char *dst,
                      unsigned char *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(half *dst,
                      half *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(short *dst,
                      short *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(unsigned short *dst,
                      unsigned short *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(float *dst,
                      float *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(int *dst,
                      int *src,
                      int height,
                      int width)
```

```
void __bang_rotate90(unsigned int *dst,
                      unsigned int *src,
                      int height,
                      int width)
```

Rotates <src> , a matrix, whose size is <height> * <width> , by 90 degrees in clockwise direction, and stores the result in <dst> .

Parameters

- [out] dst: The address of destination matrix.
- [in] src: The address of source matrix.
- [in] height: The height of <src>.
- [in] width: The width of <src>.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <height> * sizeof(type) and <width> * sizeof(type) must be divisible by 128 bytes on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> cannot be overlapped with <src>;
- <height> and <width> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define WIDTH 128
#define HEIGHT 64
#define LEN (HEIGHT * WIDTH)

__mlu_entry__ void kernel(unsigned char* dst, unsigned char* src) {
    __nram__ unsigned char ny[LEN];
    __nram__ unsigned char nx[LEN];
    __memcpy(nx, src, LEN * sizeof(unsigned char), GDRAM2NRAM);
    __bang_rotate90(ny, nx, HEIGHT, WIDTH);
    __memcpy(dst, ny, LEN * sizeof(unsigned char), NRAM2GDRAM);
```

```
}
```

3.18.9 __bang_select

```
void __bang_select(half *dst,
                    half *src,
                    half *index,
                    int elem_count)
```

```
void __bang_select(bfloat16_t *dst,
                    bfloat16_t *src,
                    bfloat16_t *index,
                    int elem_count)
```

```
void __bang_select(float *dst,
                    float *src,
                    float *index,
                    int elem_count)
```

Selects elements in one vector according to the corresponding values in another vector. The elements in `<src>` will be selected if corresponding elements in `<index>` are not equal to zero. The result is composed of three parts. The first 4-byte is the number of selected element, whose data type is `unsigned int`. The next 124-byte is zero. The rest bytes are the selected elements.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `index`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<dst>` can be overlapped with `<src>`;
- The `<src>`, `<index>` and `<dst>` must point to `_nram_` address space;
- `<elem_count> * sizeof(type)` must be a multiple of 128 bytes;
- The address of `<src>`, `<index>` and `<dst>` must be 64-byte aligned;
- If the total number of bytes of the selected elements is not a multiple of 128, it will be aligned to 128 bytes, and the padding bytes will be set to zero;
- The reserved space for selected data in `<dst>` cannot be smaller than the size of `<src>` and if the selected data does not fill the entire `<dst>` space, the remaining data in `<dst>` may also be changed.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.10 __bang_select_bitindex

```
void __bang_select_bitindex(half *dst,
                            half *src,
                            void *bitindex,
                            int elem_count)
```

```
void __bang_select_bitindex(bfloat16_t *dst,
                           bfloat16_t *src,
                           void *bitindex,
                           int elem_count)
```

```
void __bang_select_bitindex(float *dst,
                           float *src,
                           void *bitindex,
                           int elem_count)
```

Cambricon@155chb

Selects elements in <src> if the corresponding bit values in <bitindex> are not equal to zero. The result is saved to <dst>, which composes of three parts. The first 4-byte is the number of selected elements, whose data type is `unsigned int`. The next 124-byte is zero. The rest bytes are the selected elements.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `bitindex`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on (m)tp_5xx or higher;
- <code>elem_count</code> must be greater than zero;
- <code>dst</code> can be overlapped with <code>src</code>;
- The <code>src</code>, <code>bitindex</code> and <code>dst</code> must point to __nram__ address space;
- <code>elem_count</code> must be divisible by 512;
- The address of <code>src</code>, <code>bitindex</code> and <code>dst</code> must be 64-byte aligned;
- If the total number of bytes of the selected elements is not a multiple of 128, it will be aligned to 128 bytes, and the padding bytes will be set to zero;
- The reserved space for selected data in <code>dst</code> cannot be smaller than the size of <code>src</code>

and if the selected data does not fill the entire <dst> space, the remaining data in <dst> may also be changed.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.11 __bang_tiling_2d_b128

```
void __bang_tiling_2d_b128(void *dst,
                           void *src,
                           int n2,
                           int s2,
                           int n1,
                           int s1,
                           int n7,
                           int s7,
                           int n6,
                           int s6)
```

Cambricon@155chb

Applies 2D tiling operation on <src> for matrix transpose and store the matrix to the <dst>.

As shown in Figure [The Process of 2 Dimensional Tiling Function](#), the cells with blue background indicate the data to be copied, and red blocks indicate basic element unit. The cells with blue background indicate 64 bytes of the data to be copied in each segment. In this case, there are 4 segments in the first dimension. <s1> of data in each segment are copied to destination area <n1> times. There are 2 segments in the second dimension. <s2> of data in each segment are copied to destination area <n2> times. Then, calculate matrix transpose based on 128-bit as one element and store the matrix into corresponding segments and dimensions of <dst>.

Parameters

- [out] dst: The address of output matrix.
- [in] src: The address of input matrix.
- [in] n2: Tiling input iteration 2.
- [in] s2: Tiling input stride 2.
- [in] n1: Tiling input iteration 1.
- [in] s1: Tiling input stride 1.
- [in] n7: Tiling output iteration 7.
- [in] s7: Tiling output stride 7.
- [in] n6: Tiling output iteration 6.
- [in] s6: Tiling output stride 6.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- $n1 \times n2 = n6 \times n7 = 4$;
- <n1>, <n2>, <n6> and <n7> must be an immediate integer;
- <s1>, <s2>, <s6> and <s7> must be greater than or equal to zero;
- Unit of <s1>, <s2>, <s6>, <s7> is 64 bytes;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.12 __bang_tiling_2d_b16

```
void __bang_tiling_2d_b16(void *dst,
                           void *src,
                           int n2,
                           int s2,
                           int n1,
                           int s1,
                           int n7,
                           int s7,
                           int n6,
                           int s6)
```

Cambricon@155chb

Applies 2D tiling operation on <src> for matrix transpose and store the matrix to the <dst>.

As shown in Figure [The Process of 2 Dimensional Tiling Function](#), the cells with blue background indicate the data to be copied, and red blocks indicate basic element unit. The cells with blue background indicate 64 bytes of the data to be copied in each segment. In this case, there are 4 segments in the first dimension. <s1> of data in each segment are copied to destination area <n1> times. There are 2 segments in the second dimension. <s2> of data in each segment are copied to destination area <n2> times. Then, calculate matrix transpose based on 16-bit as one element and store the matrix into corresponding segments and dimensions of <dst>.

Parameters

- [out] dst: The address of output matrix.
- [in] src: The address of input matrix.
- [in] n2: Tiling input iteration 2.
- [in] s2: Tiling input stride 2.

- [in] n1: Tiling input iteration 1.
- [in] s1: Tiling input stride 1.
- [in] n7: Tiling output iteration 7.
- [in] s7: Tiling output stride 7.
- [in] n6: Tiling output iteration 6.
- [in] s6: Tiling output stride 6.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- $n1 \times n2 = n6 \times n7 = 32$;
- <n1>, <n2>, <n6> and <n7> must be an immediate integer;
- <s1>, <s2>, <s6> and <s7> must be greater than or equal to zero;
- Unit of <s1>, <s2>, <s6>, <s7> is 64 bytes;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.13 __bang_tiling_2d_b256

```
void __bang_tiling_2d_b256(void *dst,
                           void *src,
                           int n2,
                           int s2,
                           int n1,
                           int s1,
                           int n7,
                           int s7,
                           int n6,
                           int s6)
```

Applies 2D tiling operation on <src> for matrix transpose and store the matrix to the <dst>.

As shown in Figure [The Process of 2 Dimensional Tiling Function](#), the cells with blue background indicate the data to be copied, and red blocks indicate basic element unit. The cells with blue background indicate 64 bytes of the data to be copied in each segment. In this case, there are 4 segments in the first dimension. <s1> of data in each segment are copied to destination area <n1> times. There are 2 segments in the second dimension. <s2> of data in each segment are copied to destination area <n2> times. Then, calculate matrix transpose

based on 256-bit as one element and store the matrix into corresponding segments and dimensions of <dst>.

Parameters

- [out] dst: The address of output matrix.
- [in] src: The address of input matrix.
- [in] n2: Tiling input iteration 2.
- [in] s2: Tiling input stride 2.
- [in] n1: Tiling input iteration 1.
- [in] s1: Tiling input stride 1.
- [in] n7: Tiling output iteration 7.
- [in] s7: Tiling output stride 7.
- [in] n6: Tiling output iteration 6.
- [in] s6: Tiling output stride 6.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <n1> × <n2> = <n6> × <n7> = 2;
- <n1>, <n2>, <n6> and <n7> must be an immediate integer;
- <s1>, <s2>, <s6> and <s7> must be greater than or equal to zero;
- Unit of <s1>, <s2>, <s6>, <s7> is 64 bytes;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.14 __bang_tiling_2d_b32

```
void __bang_tiling_2d_b32(void *dst,
                           void *src,
                           int n2,
                           int s2,
                           int n1,
                           int s1,
                           int n7,
                           int s7,
                           int n6,
                           int s6)
```

Applies 2D tiling operation on `<src>` for matrix transpose and store the matrix to the `<dst>`.

As shown in Figure [The Process of 2 Dimensional Tiling Function](#), the cells with blue background indicate the data to be copied, and red blocks indicate basic element unit. The cells with blue background indicate 64 bytes of the data to be copied in each segment. In this case, there are 4 segments in the first dimension. `<s1>` of data in each segment are copied to destination area `<n1>` times. There are 2 segments in the second dimension. `<s2>` of data in each segment are copied to destination area `<n2>` times. Then, calculate matrix transpose based on 32-bit as one element and store the matrix into corresponding segments and dimensions of `<dst>`.

Parameters

- [out] `dst`: The address of output matrix.
- [in] `src`: The address of input matrix.
- [in] `n2`: Tiling input iteration 2.
- [in] `s2`: Tiling input stride 2.
- [in] `n1`: Tiling input iteration 1.
- [in] `s1`: Tiling input stride 1.
- [in] `n7`: Tiling output iteration 7.
- [in] `s7`: Tiling output stride 7.
- [in] `n6`: Tiling output iteration 6.
- [in] `s6`: Tiling output stride 6.

Return

- `void`.

Remark

Cambricon@155chb

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- $<n1> \times <n2> = <n6> \times <n7> = 16$;
- `<n1>`, `<n2>`, `<n6>` and `<n7>` must be an immediate integer;
- `<s1>`, `<s2>`, `<s6>` and `<s7>` must be greater than or equal to zero;
- Unit of `<s1>`, `<s2>`, `<s6>`, `<s7>` is 64 bytes;
- `<dst>` cannot be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__` ≥ 200 ;
- CNCC Version: `cncc --version` $\geq 2.8.0$;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` $\geq \text{compute_20}$;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` $\geq (\text{m})\text{tp_2xx}$.

Example

- None.

3.18.15 __bang_tiling_2d_b64

```
void __bang_tiling_2d_b64(void *dst,
                           void *src,
                           int n2,
                           int s2,
                           int n1,
                           int s1,
                           int n7,
                           int s7,
                           int n6,
                           int s6)
```

Applies 2D tiling operation on <src> for matrix transpose and store the matrix to the <dst>.

As shown in Figure [The Process of 2 Dimensional Tiling Function](#), the cells with blue background indicate the data to be copied, and red blocks indicate basic element unit. The cells with blue background indicate 64 bytes of the data to be copied in each segment. In this case, there are 4 segments in the first dimension. <s1> of data in each segment are copied to destination area <n1> times. There are 2 segments in the second dimension. <s2> of data in each segment are copied to destination area <n2> times. Then, calculate matrix transpose based on 64-bit as one element and store the matrix into corresponding segments and dimensions of <dst>.

Parameters

- [out] dst: The address of output matrix.
- [in] src: The address of input matrix.
- [in] n2: Tiling input iteration 2.
- [in] s2: Tiling input stride 2.
- [in] n1: Tiling input iteration 1.
- [in] s1: Tiling input stride 1.
- [in] n7: Tiling output iteration 7.
- [in] s7: Tiling output stride 7.
- [in] n6: Tiling output iteration 6.
- [in] s6: Tiling output stride 6.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <n1> × <n2> = <n6> × <n7> = 8;
- <n1>, <n2>, <n6> and <n7> must be an immediate integer;
- <s1>, <s2>, <s6> and <s7> must be greater than or equal to zero;
- Unit of <s1>, <s2>, <s6>, <s7> is 64 bytes;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.18.16 `__bang_tiling_2d_b8`

```
void __bang_tiling_2d_b8(void *dst,
                         void *src,
                         int n2,
                         int s2,
                         int n1,
                         int s1,
                         int n7,
                         int s7,
                         int n6,
                         int s6)
```

Applies 2D tiling operation on `<src>` for matrix transpose and stores the matrix to the `<dst>`.

As shown in Figure [The Process of 2 Dimensional Tiling Function](#), the cells with blue background indicate the data to be copied, and red blocks indicate basic element unit. The cells with blue background indicate 64 bytes of the data to be copied in each segment. In this case, there are 4 segments in the first dimension. `<s1>` of data in each segment are copied to destination area `<n1>` times. There are 2 segments in the second dimension. `<s2>` of data in each segment are copied to destination area `<n2>` times. Then, calculate matrix transpose based on 8-bit as one element and store the matrix into corresponding segments and dimensions of `<dst>`.

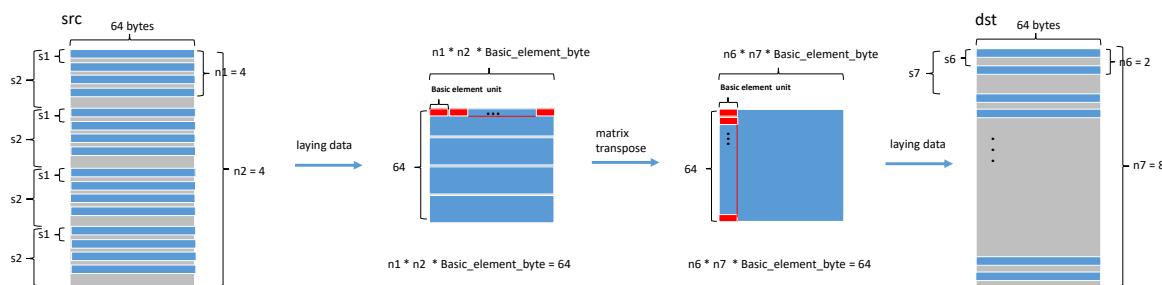


Fig. 3.13: The Process of 2 Dimensional Tiling Function

Parameters

- [out] `dst`: The address of output matrix.
- [in] `src`: The address of input matrix.
- [in] `n2`: Tiling input iteration 2.
- [in] `s2`: Tiling input stride 2.

- [in] n1: Tiling input iteration 1.
- [in] s1: Tiling input stride 1.
- [in] n7: Tiling output iteration 7.
- [in] s7: Tiling output stride 7.
- [in] n6: Tiling output iteration 6.
- [in] s6: Tiling output stride 6.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- $n1 \times n2 = n6 \times n7 = 64$;
- <n1>, <n2>, <n6> and <n7> must be an immediate integer;
- <s1>, <s2>, <s6> and <s7> must be greater than or equal to zero;
- Unit of <s1>, <s2>, <s6>, <s7> is 64 bytes;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.18.17 __bang_tiling_3d_b1024

```
void __bang_tiling_3d_b1024(void *dst,
                           void *src,
                           int n5,
                           int s5,
                           int n4,
                           int s4,
                           int n3,
                           int s3,
                           int n10,
                           int s10,
                           int n9,
                           int s9,
                           int n8,
                           int s8,
                           int op)
```

Copies data from <src> to <dst> in 3 dimensions.

As shown in Figure [The Process of 3 Dimensional Tiling Function](#), the cells with blue

background indicate $\langle op \rangle$ of the data to be copied in each segment. In this case, there are 3 segments in the first dimension. $\langle s3 \rangle$ of data in each segment are copied to destination area $\langle n3 \rangle$ times. There are 2 segments in the second dimension. $\langle s4 \rangle$ of data in each segment are copied to destination area $\langle n4 \rangle$ times. And there are 2 segments in the third dimension. $\langle s5 \rangle$ of data in each segment are copied to destination area $\langle n5 \rangle$ times. Then, copy the data into corresponding segments and dimensions of $\langle dst \rangle$.

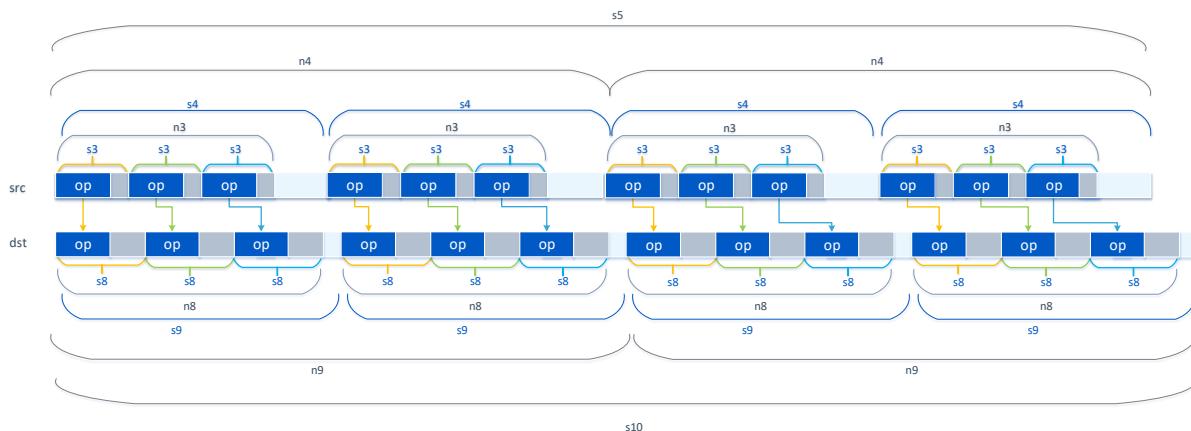


Fig. 3.14: The Process of 3 Dimensional Tiling Function

Parameters

- [out] $\langle dst \rangle$: The address of output tensor.
- [in] $\langle src \rangle$: The address of input tensor.
- [in] $\langle n5 \rangle$: Tiling input outer iteration 5.
- [in] $\langle s5 \rangle$: Tiling input outer stride 5.
- [in] $\langle n4 \rangle$: Tiling input outer iteration 4.
- [in] $\langle s4 \rangle$: Tiling input outer stride 4.
- [in] $\langle n3 \rangle$: Tiling input outer iteration 3.
- [in] $\langle s3 \rangle$: Tiling input outer stride 3.
- [in] $\langle n10 \rangle$: Tiling output outer iteration 10.
- [in] $\langle s10 \rangle$: Tiling output outer stride 10.
- [in] $\langle n9 \rangle$: Tiling output outer iteration 9.
- [in] $\langle s9 \rangle$: Tiling output outer stride 9.
- [in] $\langle n8 \rangle$: Tiling output outer iteration 8.
- [in] $\langle s8 \rangle$: Tiling output outer stride 8.
- [in] $\langle op \rangle$: The number of tiling unit (128 bytes).

Return

- void.

Remark

- $\langle src \rangle$ and $\langle dst \rangle$ must point to `__nram__` address space;
- $\langle op \rangle$ must be an immediate integer;
- The address of $\langle src \rangle$ and $\langle dst \rangle$ must be 64-byte aligned on `(m)tp_2xx`;
- Unit of $\langle s3 \rangle$, $\langle s4 \rangle$, $\langle s5 \rangle$, $\langle s8 \rangle$, $\langle s9 \rangle$ and $\langle s10 \rangle$ is 64 bytes;
- $\langle s3 \rangle$, $\langle s4 \rangle$, $\langle s5 \rangle$, $\langle s8 \rangle$, $\langle s9 \rangle$ and $\langle s10 \rangle$ must be greater than or equal to zero;

- $< n3 > \times < n4 > \times < n5 > == < n8 > \times < n9 > \times < n10 >;$
- $\langle dst \rangle$ cannot be overlapped with $\langle src \rangle$.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

3.18.18 `__bang_transpose`

```
void __bang_transpose(half *dst,
                      half *src,
                      int height,
                      int width)
```

```
void __bang_transpose(short *dst,
                      short *src,
                      int height,
                      int width)
```

```
void __bang_transpose(unsigned short *dst,
                      unsigned short *src,
                      int height,
                      int width)
```

```
void __bang_transpose(int8_t *dst,
                      int8_t *src,
                      int height,
                      int width)
```

```
void __bang_transpose(char *dst,
                      char *src,
                      int height,
                      int width)
```

```
void __bang_transpose(unsigned char *dst,
                      unsigned char *src,
                      int height,
                      int width)
```

```
void __bang_transpose(float *dst,
                      float *src,
                      int height,
                      int width)
```

```
void __bang_transpose(int *dst,
                      int *src,
                      int height,
                      int width)
```

```
void __bang_transpose(unsigned int *dst,
                      unsigned int *src,
                      int height,
                      int width)
```

Transpose operand `src[height][width]`, a matrix, to `dst[width][height]`.

Parameters

- [out] `dst`: The address of destination matrix, and the matrix has WH data layout.
- [in] `src`: The address of source matrix, and the matrix has HW data layout.
- [in] `height`: The height of `<src>`.
- [in] `width`: The width of `<src>`.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<height> * sizeof(type)` and `<width> * sizeof(type)` must be divisible by 64 on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` cannot be overlapped with `<src>`;
- `<height>` and `<width>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define WIDTH 128
#define HEIGHT 64
#define LEN (HEIGHT * WIDTH)

__mlu_entry__ void kernel(short* dst, short* src) {
    __nram__ short ny[LEN];
```

```

__nram__ short nx[LEN];
__memcpy(nx, src, LEN * sizeof(short), GDRAM2NRAM);
__bang_transpose(ny, nx, HEIGHT, WIDTH);
__memcpy(dst, ny, LEN * sizeof(short), NRAM2GDRAM);
}

```

3.19 Vector Operation Functions

3.19.1 __bang_abs

```
void __bang_abs(bfloat16_t *dst,
                 bfloat16_t *src,
                 int elem_count)
```

```
void __bang_abs(half *dst,
                 half *src,
                 int elem_count)
```

```
void __bang_abs(float *dst,
                 float *src,
                 int elem_count)
```

```
void __bang_abs(char *dst,
                 char *src,
                 int elem_count)
```

```
void __bang_abs(short *dst,
                 short *src,
                 int elem_count)
```

```
void __bang_abs(int *dst,
                 int *src,
                 int elem_count)
```

This function computes the absolute value of <src> element-wisely and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;

- <elem_count> must be greater than zero;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_322.

Example

- None.

3.19.2 __bang_add

```
void __bang_add(int *dst,
                int *src0,
                int *src1,
                int elem_count)
```

```
void __bang_add(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)
```

```
void __bang_add(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)
```

```
void __bang_add(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_add(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_add(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function performs addition operation element-wisely on <src0> and <src1> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation](#)

Functions for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- float is supported on (m)tp_2xx or higher;
- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_add(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.19.3 __bang_add_scalar

```
void __bang_add_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)

void __bang_add_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)

void __bang_add_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)

void __bang_add_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_add_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)

void __bang_add_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function adds `<value>` to `<elem_count>` elements of `<src>` and saves the result in `<dst>`.

See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src>` can be overlapped with `<dst>`;

- `<elem_count> * sizeof(type)` must be divisible by 128 on (m)tp_2xx;
- `<elem_count>` must be greater than zero;
- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_add_scalar(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

Cambricon@155chb

3.19.4 __bang_add_tz

```
void __bang_add_tz(half *dst,
                    half *src0,
                    half *src1,
                    int elem_count)
```

```
void __bang_add_tz(float *dst,
                    float *src0,
                    float *src1,
                    int elem_count)
```

This function performs addition operation element-wisely on `<src0>` and `<src1>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_add_tz(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.19.5 __bang_adds

```
void __bang_adds(unsigned char *dst,
                 unsigned char *src0,
                 unsigned char *src1,
                 int elem_count)
```

```
void __bang_adds(unsigned short *dst,
                  unsigned short *src0,
                  unsigned short *src1,
                  int elem_count)
```

```
void __bang_adds(unsigned int *dst,
                  unsigned int *src0,
                  unsigned int *src1,
                  int elem_count)
```

```
void __bang_adds(char *dst,
                  char *src0,
                  char *src1,
                  int elem_count)
```

```
void __bang_adds(short *dst,
                  short *src0,
                  short *src1,
                  int elem_count)
```

```
void __bang_adds(int *dst,
                  int *src0,
                  int *src1,
                  int elem_count)
```

This function performs saturated addition operation element-wisely on <src0> and <src1> and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.6 __bang_adds_scalar

```
void __bang_adds_scalar(char *dst,
                        char *src,
                        char value,
                        int elem_count)

void __bang_adds_scalar(short *dst,
                        short *src,
                        short value,
                        int elem_count)

void __bang_adds_scalar(int *dst,
                        int *src,
                        int value,
                        int elem_count)
```

This function adds <value> to <elem_count> elements of <src> and saves the saturated result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.7 __bang_count

```
unsigned int __bang_count(half *src,
                           int elem_count)
```

```
unsigned int __bang_count(bfloat16_t *src,
                           int elem_count)
```

```
unsigned int __bang_count(float *src,
                           int elem_count)
```

Counts the number of non-zero elements in the input vector.

Parameters

- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `unsigned int` The number of non-zero elements in the input vector.

Remark

- `<elem_count>` must be greater than zero;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src>` must point to `__nram__` address space;
- The address of `<src>` must be 64-byte aligned on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(float* src, int elem_count) {
    unsigned int counter = __bang_count(src, elem_count);
}
```

3.19.8 __bang_count_bitindex

```
unsigned int __bang_count_bitindex(half *src,
                                  int elem_count)
```

```
unsigned int __bang_count_bitindex(bfloat16_t *src,
                                  int elem_count)
```

```
unsigned int __bang_count_bitindex(float *src,
                                  int elem_count)
```

Counts the number of non-zero bit in the input vector.

Parameters

- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `unsigned int` The number of non-zero bit in the input vector.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src>` must point to `__nram__` address space;
- The address of `<src>` must be 64-byte aligned on `(m)tp_2xx`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(float* src, int elem_count) {
    unsigned int counter = __bang_count_bitindex(src, elem_count);
}
```

3.19.9 __bang_cycle_add

```
void __bang_cycle_add(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_add(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_add(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_add(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_add(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_add(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = \lceil \frac{\text{src_elem_count}}{\text{seg_elem_count}} \rceil$), adds each element in each part of `<src>` and the corresponding element in `<seg>`, and assigns the result to `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

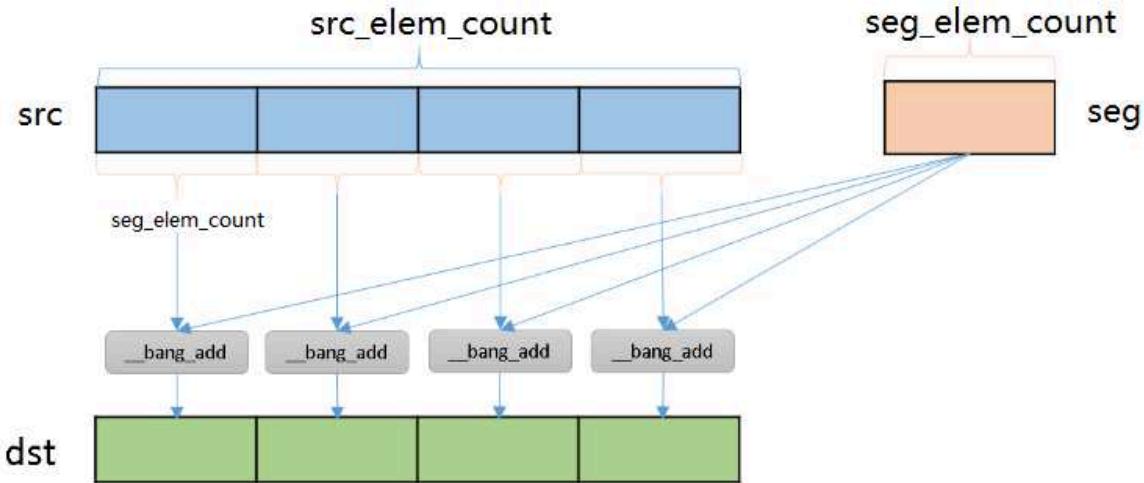


Fig. 3.15: Description of __bang_cycle_add Operation

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in <src> vector.
- [in] `seg_elem_count`: The number of elements in <seg> vector.

Return

- `void`.

Remark

- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define N1 256
#define N2 65280
```

```
--mlu_entry__ void kernel(half* c, half* a, half* b) {
    __nram__ half a_tmp[N1];
    __nram__ half b_tmp[N2];
    __nram__ half c_tmp[N2];
    __memcpy(a_tmp, a, N1 * sizeof(half), GDRAM2NRAM);
    __memcpy(b_tmp, b, N2 * sizeof(half), GDRAM2NRAM);
    __bang_cycle_add(c_tmp, b_tmp, a_tmp, N2, N1);
    __memcpy(c, c_tmp, N2 * sizeof(half), NRAM2GDRAM);
}
```

3.19.10 __bang_cycle_add_tz

```
void __bang_cycle_add_tz(half *dst,
                         half *src,
                         half *seg,
                         int src_elem_count,
                         int seg_elem_count)
```

```
void __bang_cycle_add_tz(float *dst,
                         float *src,
                         float *seg,
                         int src_elem_count,
                         int seg_elem_count)
```

Cambricon@155chb

Adds two input vectors segment by segment in round-to-zero mode. This function is calculated in the same way as ::__bang_cycle_add function. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <src_elem_count> * sizeof(type) and <seg_elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- <src>, <seg> and <dst> must point to __nram__ address space;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.6.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define N1 256
#define N2 65280

__mlu_entry__ void kernel(half* c, half* a, half* b) {
    __nram__ half a_tmp[N1];
    __nram__ half b_tmp[N2];
    __nram__ half c_tmp[N2];
    __memcpy(a_tmp, a, N1 * sizeof(half), GDRAM2NRAM);
    __memcpy(b_tmp, b, N2 * sizeof(half), GDRAM2NRAM);
    __bang_cycle_add_tz(c_tmp, b_tmp, a_tmp, N2, N1);
    __memcpy(c, c_tmp, N2 * sizeof(half), NRAM2GDRAM);
}
```

3.19.11 __bang_cycle_adds

```
void __bang_cycle_adds(unsigned int *dst,
                      unsigned int *src,
                      unsigned int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_adds(unsigned short *dst,
                      unsigned short *src,
                      unsigned short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_adds(unsigned char *dst,
                      unsigned char *src,
                      unsigned char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_adds(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_adds(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_adds(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = <\text{src_elem_count}> / <\text{seg_elem_count}>$), adds each element in each part of `<src>` and the corresponding element in `<seg>`, and assigns the saturated result to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

- None.

3.19.12 __bang_cycle_mul

```
void __bang_cycle_mul(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_mul(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_mul(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_mul(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_mul(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_mul(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = <\text{src_elem_count}> / <\text{seg_elem_count}>$), multiplies each element in each part of `<src>` and the corresponding element in `<seg>`, and assigns the result to `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.

- [in] `src_elem_count`: The number of elements in <src> vector.
- [in] `seg_elem_count`: The number of elements in <seg> vector.

Return

- void.

Remark

- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <src_elem_count> * sizeof(type) and <seg_elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- <src>, <seg> and <dst> must point to __nram__ address space;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

Cambricon@155chb

3.19.13 __bang_cycle_mulh

```
void __bang_cycle_mulh(unsigned int *dst,
                      unsigned int *src,
                      unsigned int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_mulh(unsigned short *dst,
                      unsigned short *src,
                      unsigned short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_mulh(unsigned char *dst,
                      unsigned char *src,
                      unsigned char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_mulh(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_mulh(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_mulh(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = \lceil \frac{\text{src_elem_count}}{\text{seg_elem_count}} \rceil$), multiplies each element in each part of `<src>` and the corresponding element in `<seg>`, and assigns the upper half of result to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

- None.

3.19.14 __bang_cycle_muls

```
void __bang_cycle_muls(unsigned short *dst,
                      unsigned char *src,
                      unsigned char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_muls(unsigned int *dst,
                      unsigned short *src,
                      unsigned short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_muls(int *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_muls(short *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = \lceil \text{src_elem_count} \rceil / \text{seg_elem_count} \rceil$), multiplies each element in each part of `<src>` and the corresponding element in `<seg>`, and assigns the result to `<dst>`. The size of `<dst>` is as long as that of `<src>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;

- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

- None.

3.19.15 __bang_cycle_sub

```
void __bang_cycle_sub(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_sub(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_sub(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_sub(half *dst,
                      half *src,
                      half *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_sub(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_sub(float *dst,
                      float *src,
                      float *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = \lfloor \frac{\text{src_elem_count}}{\text{seg_elem_count}} \rfloor$), subtracts each element in each part of `<src>` and the corresponding element in `<seg>`, and

assigns the result to `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `int` and `short` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<src_elem_count> * sizeof(type)` and `<seg_elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

3.19.16 `__bang_cycle_subs`

```
void __bang_cycle_subs(unsigned int *dst,
                      unsigned int *src,
                      unsigned int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_subs(unsigned short *dst,
                      unsigned short *src,
                      unsigned short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_subs(unsigned char *dst,
                      unsigned char *src,
                      unsigned char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_subs(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_subs(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_subs(char *dst,
                      char *src,
                      char *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

This function divides `<src>` into N parts ($N = \lfloor \frac{\text{src_elem_count}}{\text{seg_elem_count}} \rfloor$), subtracts each element in each part of `<src>` and the corresponding element in `<seg>`, and assigns the saturated result to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;

- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.19.17 __bang_findfirst1

```
unsigned int __bang_findfirst1(half *src,
                               int elem_count)
```

```
unsigned int __bang_findfirst1(bfloat16_t *src,
                               int elem_count)
```

```
unsigned int __bang_findfirst1(float *src,
                               int elem_count)
```

Finds the first non-zero data in the values of <src>, and returns the index of the first non-zero data. If <src> is all zero, returns 0xffff-ffff-ffff-ffff.

Parameters

- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- unsigned int.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero;
- <elem_count> * sizeof (type) must be divisible by 128 on (m)tp_2xx;
- <src> must point to __nram__ address space;
- The address of <src> must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(half* a) {
    __nram__ half a_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    unsigned int num = __bang_findfirst1(a_tmp, DATA_SIZE);
}
```

3.19.18 __bang_findlast1

```
unsigned int __bang_findlast1(half *src,
                             int elem_count)
```

```
unsigned int __bang_findlast1(bfloat16_t *src,
                             int elem_count)
```

```
unsigned int __bang_findlast1(float *src,
                             int elem_count)
```

Finds the last non-zero data in the values of <src>, and returns the index of the last non-zero data. If <src> is all zero, returns 0xffff-ffff-ffff-ffff.

Parameters

- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- unsigned int.

Remark

- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero;
- <elem_count> * sizeof (type) must be divisible by 128 on (m)tp_2xx;
- <src> must point to _nram_ address space;
- The address of <src> must be 64-byte aligned on (m)tp_2xx;

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.19.19 __bang_floor

```
void __bang_floor(float *dst,
                  float *src,
                  int elem_count)
```

This function performs floor operation element-wisely on <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the source vector.

- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <dst> cannot be overlapped with <src>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.19.20 __bang_histogram

```
void __bang_histogram(half *dst,
                      int8_t *src,
                      int8_t *kernel,
                      int size)
```

Cambricon@155chb

```
void __bang_histogram(float *dst,
                      int8_t *src,
                      int8_t *kernel,
                      int size)
```

```
void __bang_histogram(int16_t *dst,
                      int8_t *src,
                      int8_t *kernel,
                      int size)
```

```
void __bang_histogram(int *dst,
                      int8_t *src,
                      int8_t *kernel,
                      int size)
```

```
void __bang_histogram(half *dst,
                      int16_t *src,
                      int16_t *kernel,
                      int size)
```

```

void __bang_histogram(float *dst,
                      int16_t *src,
                      int16_t *kernel,
                      int size)

void __bang_histogram(int16_t *dst,
                      int16_t *src,
                      int16_t *kernel,
                      int size)

void __bang_histogram(int *dst,
                      int16_t *src,
                      int16_t *kernel,
                      int size)

void __bang_histogram(float *dst,
                      half *src,
                      half *kernel,
                      int size)

void __bang_histogram(float *dst,
                      bfloat16_t *src,
                      bfloat16_t *kernel,
                      int size)

```

Generate a histogram of <src> according to kernel. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector
- [in] `src`: The address of source vector
- [in] `kernel`: The address of kernel vector
- [in] `size`: The elements number of source vector

Return

- `void`.

Remark

- <dst> and <src> must point to `__nram__` address space;
- <kernel> must point to `__wram__` address space;
- The address of <dst> and <src> must be 64-byte aligned;
- The address of <kernel> must be 32-byte aligned;
- `<size> * sizeof(typeof<src>)` must be 64-byte aligned on `(m)tp_2xx`;
- The element number of <dst> is 64;
- <size> must be greater than 0;
- <dst> cannot be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 220` except 270 and 290;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20` excepts `(m)tp_270` and `(m)tp_5xx`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_220` excepts `(m)tp_270` and `(m)tp_5xx`.

Compatibility between Various Architectures

Table 3.41: Histogram Data Types Supported on `(m)tp_220`

Src Type	Kernel Type	Dst Type
int8	int8	int32
int16	int16	int32

Table 3.42: Histogram Data Types Supported on `mtp_372`

Src Type	Kernel Type	Dst Type
int8	int8	half
int8	int8	float
int8	int8	int16
int8	int8	int32
int16	int16	half
int16	int16	float
int16	int16	int16
int16	int16	int32
half	half	float
bfloat16_t	bfloat16_t	float

Table 3.43: Histogram Data Types Supported on tp_322

Src Type	Kernel Type	Dst Type
int8	int8	half
int8	int8	float
int8	int8	int16
int8	int8	int32
int16	int16	half
int16	int16	float
int16	int16	int16
int16	int16	int32

Table 3.44: Histogram Data Types Supported on (m)tp_5xx

Src Type	Kernel Type	Dst Type
int8	int8	half
int8	int8	float
int8	int8	int16
int8	int8	int32
half	half	float
bfloat16_t	bfloat16_t	float

Example

The data layout of <kernel> is 64 rows. Every row is 64 bytes, and the value of each row is corresponding to the histogram of x-axis. There are 32 values in each row when data type is int16_t, and 64 values when data type is int8_t;

For example, src is 0 1 2 3 0 1 2 3 3. As kernel's first line is 0, this function selects <src>'s "0" and generates 2; as kernel's second line is 1, it selects <src>'s "1" and generates 2; as kernel's third line is 2, it selects <src>'s "2" and generates 2; as kernel's fourth line is 3, it selects <src>'s "3" and generates 3; as kernel's the other line is 0, it generates 2. Therefore, <dst> is 2 2 2 3 2 2 ... 2 and the number of elements of <dst> is 64.

```
#include <bang.h>

#define SRC_NUM 512
#define DST_NUM 64
#define WEI_NUM (64 * 32)

__mlu_entry__ void kernel(int32_t* dst, int16_t* src, int16_t* filter, int size) {
```

```

__nram__ int32_t dst_tmp[DST_NUM];
__nram__ int16_t src_tmp[SRC_NUM];
__wram__ int16_t filter_tmp[WEI_NUM];
__memcpy(src_tmp, src, SRC_NUM * sizeof(int16_t), GDRAM2NRAM);
__memcpy(filter_tmp, filter, WEI_NUM * sizeof(int16_t), GDRAM2WRAM);
__bang_write_zero((half*)dst_tmp, DST_NUM * 2);
__bang_histogram(dst_tmp, src_tmp, filter_tmp, size);
__memcpy(dst, dst_tmp, DST_NUM * sizeof(int32_t), NRAM2GDRAM);
}

```

3.19.21 __bang_integral

```
void __bang_integral(int *dst,
                     short *src,
                     short *kernel,
                     int size)
```

```
void __bang_integral(short *dst,
                     short *src,
                     short *kernel,
                     int size)
```

```
void __bang_integral(half *dst,
                     int16_t *src,
                     int16_t *kernel,
                     int size,
                     int fix_position)
```

```
void __bang_integral(float *dst,
                     int16_t *src,
                     int16_t *kernel,
                     int size,
                     int fix_position)
```

```
void __bang_integral(int *dst,
                     char *src,
                     char *kernel,
                     int size)
```

```
void __bang_integral(short *dst,
                     char *src,
                     char *kernel,
                     int size)
```

```
void __bang_integral(half *dst,
                     int8_t *src,
                     int8_t *kernel,
                     int size,
                     int fix_position)
```

```
void __bang_integral(float *dst,
                     int8_t *src,
                     int8_t *kernel,
                     int size,
                     int fix_position)
```

```
void __bang_integral(float *dst,
                     float *src,
                     float *kernel,
                     int size)
```

```
void __bang_integral(float *dst,
                     half *src,
                     half *kernel,
                     int size)
```

Generate integral of <src> according to <kernel>.

Parameters

- [out] dst: The address of destination vector
- [in] src: The address of source vector
- [in] kernel: The address of kernel vector
- [in] size: The elements number of source vector
- [in] fix_position: Sum of scale factor of <src> and <kernel>.

Return

- void.

Remark

- tp_322 does not support <src> whose data type is float or half;
- <dst> and <src> must point to __nram__ address space;
- <kernel> must point to __wram__ address space;
- The address of <dst> and <src> must be 64-byte aligned;
- (m)tp_5xx dose not support <src> whose data type is int16_t or short;
- The address of <kernel> must be 32-byte aligned;
- <size> * sizeof(typeof<src>) must be 64-byte aligned;
- <dst> cannot be overlapped with <src>;
- When data type of <src> is int8_t or char, <kernel> should be 64 × 64 Lower Triangular Matrix(LTM); when data type of <src> is int16_t, short or half, <kernel> should be 32 × 32 LTM; when datatype of <src> is float, <kernel> should be 16 × 16 LTM; kernel always requires 4096 bytes of memory, LTM should be placed at the beginning of the memory;
- Generally, the value of elements in the LTM should be 1, so <dst>[n] is sum of <src>[0] to <src>[n];

- When data type of <src> is `int8_t` or `char`, <size> cannot be greater than 65472; when data type of <src> is `int16_t`, `short` or `half`, <size> cannot be greater than 32736; when datatype of <src> is `float`, <size> cannot be greater than 16368;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.2.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

If <size> is 64, <src> is [1, 1, ..., 1, 1], <dst> should be [1, 2, 3, ..., 63, 64].

```
#include <bang.h>

#define DATA_NUM 64
#define KERNEL_SIZE (4096 / sizeof(short))

__mlu_entry__ void kernel(int* dst, short* src, short* kernel, int size) {
    __nram__ int dst_tmp[DATA_NUM];
    __nram__ short src_tmp[DATA_NUM];
    __wram__ short kernel_tmp[KERNEL_SIZE];
    __memcpy(src_tmp, src, DATA_NUM * sizeof(short), GDRAM2NRAM);
    __memcpy(kernel_tmp, kernel, KERNEL_SIZE * sizeof(short), GDRAM2WRAM);
    __bang_integral(dst_tmp, src_tmp, kernel_tmp, size);
    __memcpy(dst, dst_tmp, DATA_NUM * sizeof(int), NRAM2GDRAM);
}
```

3.19.22 __bang_lut_s16

```
void __bang_lut_s16(int16_t *dst,
                     int16_t *src,
                     int16_t *table,
                     int elem_count,
                     int table_length)
```

Applies lookup-table operation on <src>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `table`: The address of table vector.
- [in] `elem_count`: Number of elements in source vector.
- [in] `table_length`: Number of elements in table vector.

Return

- `void`.

Remark

- <src>, <table> and <dst> must point to __nram__ address space;
- The address of <src>, <table> and <dst> must be 64-byte aligned on (m)tp_220;
- The data type of <dst> and <table> can be any type with the same size as that of <src>;
- <elem_count> and <table_length> must be greater than zero;
- <elem_count> must be divisible by 64 on (m)tp_220;
- <dst>, <table> and <src> cannot be overlapped;
- <table_length> must be an immediate integer and divisible by 64.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 220 excepts 270 and 290;
- CNCC Version: cncc --version >= 3.2.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20 excepts (m)tp_270 and mtp_290;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_220 excepts (m)tp_270 and mtp_290.

Example

- None.

3.19.23 __bang_lut_s32

```
void __bang_lut_s32(int *dst,
                     int *src,
                     int *table,
                     int elem_count,
                     int table_length)
```

Applies lookup-table operation on <src>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] table: The address of table vector.
- [in] elem_count: Number of elements in source vector.
- [in] table_length: Number of elements in table vector.

Return

- void.

Remark

- <src>, <table> and <dst> must point to __nram__ address space;
- The address of <src>, <table> and <dst> must be 64-byte aligned on (m)tp_220;
- The data type of <dst> and <table> can be any type with the same size as that of <src>;
- <elem_count> and <table_length> must be greater than zero;
- <elem_count> must be divisible by 64 on (m)tp_220;
- <dst>, <table> and <src> cannot be overlapped;
- <table_length> must be an immediate integer and divisible by 64.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 220` excepts 270 and 290;
- CNCC Version: `cncc --version >= 3.2.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20` excepts `(m)tp_270` and `mtp_290`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_220` excepts `(m)tp_270` and `mtp_290`.

Example

- None.

3.19.24 __bang_mul

```
void __bang_mul(int *dst,
                int *src0,
                int *src1,
                int elem_count)

void __bang_mul(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)

void __bang_mul(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)

void __bang_mul(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)

void __bang_mul(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)

void __bang_mul(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function performs multiplication operation element-wisely on `<src0>` and `<src1>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of the first source vector.
- [in] `src1`: The address of the second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src0>`, `<src1>` and `<dst>` must point to `__nram__` address space;
- The address of `<src0>`, `<src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>`, `<src0>` and `<src1>` can be overlapped;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `int`, `short` and `char` are supported on `(m)tp_3xx` or higher;
- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_add](#) for more details.

3.19.25 __bang_mul_scalar

```
void __bang_mul_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_mul_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_mul_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_mul_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_mul_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_mul_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function multiplies `<elem_count>` elements of `<src>` by `<value>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `int, short` and `char` are supported on `(m)tp_3xx` or higher;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_add_scalar](#) for more details.

3.19.26 __bang_mulh

```
void __bang_mulh(unsigned int *dst,
                  unsigned int *src0,
                  unsigned int *src1,
                  int elem_count)
```

```
void __bang_mulh(unsigned short *dst,
                  unsigned short *src0,
                  unsigned short *src1,
                  int elem_count)
```

```
void __bang_mulh(unsigned char *dst,
                  unsigned char *src0,
                  unsigned char *src1,
                  int elem_count)
```

```
void __bang_mulh(char *dst,
                  char *src0,
                  char *src1,
                  int elem_count)
```

```
void __bang_mulh(short *dst,
                  short *src0,
                  short *src1,
                  int elem_count)
```

```
void __bang_mulh(int *dst,
                  int *src0,
                  int *src1,
                  int elem_count)
```

This function performs multiplication operation element-wisely on <src0> and <src1> and saves the result in <dst>. Only upper half of each result is stored in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- None.

3.19.27 __bang_mulh_scalar

```
void __bang_mulh_scalar(char *dst,
                        char *src,
                        char value,
                        int elem_count)

void __bang_mulh_scalar(short *dst,
                        short *src,
                        short value,
                        int elem_count)

void __bang_mulh_scalar(int *dst,
                        int *src,
                        int value,
                        int elem_count)
```

This function multiplies <value> by <elem_count> elements of <src> and saves the result in <dst>. Only the upper half of each result is stored in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- None.

3.19.28 __bang_muls

```
void __bang_muls(unsigned short *dst,
                  unsigned char *src0,
                  unsigned char *src1,
                  int elem_count)
```

```
void __bang_muls(unsigned int *dst,
                  unsigned short *src0,
                  unsigned short *src1,
                  int elem_count)
```

```
void __bang_muls(short *dst,
                  char *src0,
                  char *src1,
                  int elem_count)
```

```
void __bang_muls(int *dst,
                  short *src0,
                  short *src1,
                  int elem_count)
```

This function performs multiplication operation element-wisely on <src0> and <src1> and saves the result in <dst>. The size of <dst> is twice as long as that of <src0> and <src1>.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.29 __bang_muls_scalar

```
void __bang_muls_scalar(char *dst,
                        char *src,
                        char value,
                        int elem_count)
```

```
void __bang_muls_scalar(short *dst,
                        short *src,
                        short value,
                        int elem_count)
```

This function multiplies `<value>` by `<elem_count>` elements of `<src>` and saves the result in `<dst>`. The size of `<dst>` is as long as that of `<src>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- `<src>` can be overlapped with `<dst>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- None.

3.19.30 __bang_nearbyint

```
void __bang_nearbyint(float *dst,
                      float *src,
                      int elem_count)
```

This function performs round operation element-wisely on <src> in round-nearest-even mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <dst> cannot be overlapped with <src>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.19.31 __bang_neg

```
void __bang_neg(char *dst,
                char *src,
                int elem_count)
```

```
void __bang_neg(short *dst,
                 short *src,
                 int elem_count)
```

```
void __bang_neg(int *dst,
                 int *src,
                 int elem_count)
```

This function computes the opposite value of <src> element-wisely and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.32 __bang_nsa

```
void __bang_nsa(char *dst,
                 char *src,
                 int elem_count)
```

```
void __bang_nsa(short *dst,
                 short *src,
                 int elem_count)
```

```
void __bang_nsa(unsigned char *dst,
                 unsigned char *src,
                 int elem_count)
```

```
void __bang_nsa(unsigned short *dst,
                 unsigned short *src,
                 int elem_count)
```

```
void __bang_nsa(unsigned int *dst,
                 unsigned int *src,
                 int elem_count)
```

```
void __bang_nsa(int *dst,
                 int *src,
                 int elem_count)
```

This function counts the leading zeros or ones in <src> element-wisely and saves the result in

`<dst>`. A leading zero is any digit that comes before the first nonzero digit in binary without the signed bit. If the value of element in `<src>` is a signed positive number, the result is the number of leading zeros minus one; if it is a negative number, the result is the number of leading ones minus one; if it is an unsigned number, the result is the number of leading zeros.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of the source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src>` can be overlapped with `<dst>`;
- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322` excepts 372;
- CNCC Version: `cncc --version >= 3.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30` excepts `mtp_372`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322` excepts `mtp_372`.

Example

- None.

Cambricon@155chb

3.19.33 __bang_popcnt

```
int __bang_popcnt(unsigned int *src,
                   int elem_count)
```

```
int __bang_popcnt(unsigned short *src,
                   int elem_count)
```

```
int __bang_popcnt(unsigned char *src,
                   int elem_count)
```

Counts the number of bits that are set to 1 in `<src>`.

Parameters

- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `int`.

Remark

- `<elem_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.4.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.19.34 __bang_rand

```
void __bang_rand(short *dst,
                  int elem_count)
```

Generates a vector of uniformly distributed random number of short type.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements in destination vector.

Return

- void.

Remark

- <elem_count> must be greater than zero and divisible by 64;
- <dst> must point to __nram__ address space;
- The address of must be 64-byte aligned on (m)tp_2xx.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(short *output) {
    __nram__ short result[DATA_SIZE];
    __bang_rand(result, DATA_SIZE);
    __memcpy(output, result, DATA_SIZE * sizeof(short), NRAM2GDRAM);
}
```

3.19.35 __bang_reduce_sum

```
void __bang_reduce_sum(half *dst,
                      half *src,
                      int elem_count)

void __bang_reduce_sum(float *dst,
                      float *src,
                      int elem_count)
```

Takes every 128-byte data from <src> to add them up and stores the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- void.

Remark

- <dst> and <src> must point to __nram__ address space;
- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src>;
- The total number of bytes of <dst> is at least 128;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The first element of every 128 bytes in destination operand <dst> is the sum of every 128-byte data, the other elements in destination operand <dst> of every 128 bytes will be set to zero on (m)tp_2xx, and they will not be changed on (m)tp_3xx;
- This instruction takes every 128 bytes to calculate each time. When vector type is float, This function will take the 32 elements to calculate. When vector type is half, This function will take the 64 elements to calculate.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

3.19.36 __bang_relu

```
void __bang_relu(char *dst,
                  char *src,
                  int elem_count)

void __bang_relu(short *dst,
                  short *src,
                  int elem_count)

void __bang_relu(int *dst,
                  int *src,
                  int elem_count)

void __bang_relu(half *dst,
                  half *src,
                  int elem_count)

void __bang_relu(bfloat16_t *dst,
                  bfloat16_t *src,
                  int elem_count)

void __bang_relu(float *dst,
                  float *src,
                  int elem_count)
```

This instruction performs relu operation on each element in vector <src> and saves the result to vector <dst>. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <dst> and <src> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;

- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

```
#include <bang.h>

#define DATA_NUM 64

__mlu_entry__ void kernel(half *dst_h, half *src_h) {
    __nram__ half nram_dst_h[DATA_NUM];
    __nram__ half nram_src_h[DATA_NUM];
    __memcpy(nram_src_h, src_h, DATA_NUM * sizeof(half), GDRAM2NRAM);
    __bang_relu(nram_dst_h, nram_src_h, DATA_NUM);
    __memcpy(dst_h, nram_dst_h, DATA_NUM * sizeof(half), NRAM2GDRAM);
}
```

3.19.37 __bang_relun

```
void __bang_relun(int *dst,
                  int *src,
                  int elem_count,
                  int nvalue)
```

```
void __bang_relun(short *dst,
                  short *src,
                  int elem_count,
                  short nvalue)
```

```
void __bang_relun(char *dst,
                  char *src,
                  int elem_count,
                  char nvalue)
```

```
void __bang_relun(half *dst,
                  half *src,
                  int elem_count,
                  half nvalue)
```

```
void __bang_relun(bfloat16_t *dst,
                  bfloat16_t *src,
                  int elem_count,
                  bfloat16_t nvalue)
```

```
void __bang_relun(float *dst,
                  float *src,
                  int elem_count,
                  float nvalue)
```

The instruction performs relun operation on each element in <src>. If the element in <src> is less than 0, corresponding element in <dst> will be 0. If the element is greater than <nvalue>, corresponding element in <dst> will be <nvalue>. Otherwise, the element in <dst> will be itself. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source.
- [in] nvalue: The N value of instruction.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <dst> and <src> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- <nvalue> must be greater than zero, and cannot be INF or NAN.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

```
#include <bang.h>

#define DATA_NUM 64

__mlu_entry__ void kernel(half *dst_h, half *src_h, half const_h) {
    __nram__ half nram_dst_h[DATA_NUM];
    __nram__ half nram_src_h[DATA_NUM];
    __memcpy(nram_src_h, src_h, DATA_NUM * sizeof(half), GDRAM2NRAM);
    __bang_relun(nram_dst_h, nram_src_h, DATA_NUM, const_h);
    __memcpy(dst_h, nram_dst_h, DATA_NUM * sizeof(half), NRAM2GDRAM);
}
```

3.19.38 __bang_rol

```
void __bang_rol(unsigned char *dst,
                unsigned char *src,
                int shift_bits,
                int elem_count)
```

```
void __bang_rol(unsigned short *dst,
                unsigned short *src,
                int shift_bits,
                int elem_count)
```

```
void __bang_rol(unsigned int *dst,
                unsigned int *src,
                int shift_bits,
                int elem_count)
```

This function performs left rotation operation on <src> element-wisely with <shift_bits> bits and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] shift_bits: The number of bits right shifted.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <shift_bits> must be greater than or equal to zero;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.39 __bang_ror

```
void __bang_ror(unsigned char *dst,
                 unsigned char *src,
                 int shift_bits,
                 int elem_count)
```

```
void __bang_ror(unsigned short *dst,
                 unsigned short *src,
                 int shift_bits,
                 int elem_count)
```

```
void __bang_ror(unsigned int *dst,
                 unsigned int *src,
                 int shift_bits,
                 int elem_count)
```

This function performs right rotation operation on <src> element-wisely with <shift_bits> bits and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the source vector.
- [in] shift_bits: The number of bits right shifted.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <shift_bits> must be greater than or equal to zero;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.40 __bang_round

```
void __bang_round(float *dst,
                  float *src,
                  int elem_count)
```

This function performs round operation element-wisely on <src> in round-nearest-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <dst> cannot be overlapped with <src>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

- None.

3.19.41 __bang_set0in32

```
void __bang_set0in32(void *dst,
                     void *src,
                     int position,
                     int elem_count)
```

Set the specified bit to 0.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] position: The specified position to be set to 0.
- [in] elem_count: Number of elements in source.

Return

- void.

Remark

- <elem_count> must be greater than zero;

- The bit width of element in <src> and <dst> is 32 bits;
- <dst> and <src> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- None.

3.19.42 __bang_set1in32

```
void __bang_set1in32(void *dst,
                      void *src,
                      int position,
                      int elem_count)
```

Set the specified bit to 1.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] position: The specified position to be set to 1.
- [in] elem_count: Number of elements in source.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> and <src> must point to __nram__ address space;
- The bit width of element in <src> and <dst> is 32 bits;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- None.

3.19.43 __bang_sll

```
void __bang_sll(char *dst,
                 char *src,
                 int shift_bits,
                 int elem_count)

void __bang_sll(short *dst,
                 short *src,
                 int shift_bits,
                 int elem_count)

void __bang_sll(int *dst,
                 int *src,
                 int shift_bits,
                 int elem_count)
```

This function performs logical left shift element-wisely on <src> with <shift_bits> bits and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the first source vector.
- [in] shift_bits: The number of bits left shifted.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <shift_bits> must be greater than or equal to zero;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_322.

Example

- None.

3.19.44 __bang_square

```
void __bang_square(unsigned short *dst,
                   unsigned char *src,
                   int elem_count)
```

```
void __bang_square(unsigned int *dst,
                   unsigned short *src,
                   int elem_count)
```

```
void __bang_square(short *dst,
                   char *src,
                   int elem_count)
```

```
void __bang_square(int *dst,
                   short *src,
                   int elem_count)
```

```
void __bang_square(half *dst,
                   half *src,
                   int elem_count)
```

```
void __bang_square(bfloat16_t *dst,
                   bfloat16_t *src,
                   int elem_count)
```

```
void __bang_square(float *dst,
                   float *src,
                   int elem_count)
```

Applies square activation operation on `<src>` element-wisely, and stores the result in `<dst>`.
 $\langle dst \rangle = \langle src \rangle^2$. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `bfloat16_t` is supported on `(m)tp_5xx` or higher;
- `<elem_count>` must be greater than zero;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(half* c, half* a) {
    __nram__ half a_tmp[DATA_SIZE];
    __nram__ half c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __bang_square(c_tmp, a_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.19.45 __bang_sra

```
void __bang_sra(char *dst,
                 char *src,
                 int shift_bits,
                 int elem_count)
```

```
void __bang_sra(short *dst,
                 short *src,
                 int shift_bits,
                 int elem_count)
```

```
void __bang_sra(int *dst,
                 int *src,
                 int shift_bits,
                 int elem_count)
```

This function performs arithmetic right shift element-wisely on `<src>` with `<shift_bits>` bits and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of the first source vector.
- [in] `shift_bits`: The number of bits right shifted.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <shift_bits> must be greater than or equal to zero;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_322.

Example

- None.

3.19.46 __bang_srl

```
void __bang_srl(unsigned char *dst,
                 unsigned char *src,
                 int shift_bits,
                 int elem_count)
```

```
void __bang_srl(unsigned short *dst,
                 unsigned short *src,
                 int shift_bits,
                 int elem_count)
```

```
void __bang_srl(unsigned int *dst,
                 unsigned int *src,
                 int shift_bits,
                 int elem_count)
```

This function performs logical right shift element-wisely on <src> with <shift_bits> bits and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of the first source vector.
- [in] shift_bits: The number of bits right shifted.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <shift_bits> must be greater than or equal to zero;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_322.`

Example

- None.

3.19.47 __bang_sub

```
void __bang_sub(int *dst,
                int *src0,
                int *src1,
                int elem_count)
```

```
void __bang_sub(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)
```

```
void __bang_sub(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)
```

```
void __bang_sub(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_sub(bfloat16_t *dst,
                 bfloat16_t *src0,
                 bfloat16_t *src1,
                 int elem_count)
```

```
void __bang_sub(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

This function performs subtraction operation element-wisely on `<src0>` and `<src1>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_add](#) for more details.

Cambricon@155chb

3.19.48 __bang_sub_scalar

```
void __bang_sub_scalar(char *dst,
                      char *src,
                      char value,
                      int elem_count)
```

```
void __bang_sub_scalar(short *dst,
                      short *src,
                      short value,
                      int elem_count)
```

```
void __bang_sub_scalar(int *dst,
                      int *src,
                      int value,
                      int elem_count)
```

```
void __bang_sub_scalar(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_sub_scalar(bfloat16_t *dst,
                      bfloat16_t *src,
                      bfloat16_t value,
                      int elem_count)
```

```
void __bang_sub_scalar(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function subtracts <value> from <elem_count> elements of <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src> can be overlapped with <dst>;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <elem_count> must be greater than zero;
- int, short and char are supported on (m)tp_3xx or higher;
- bfloat16_t is supported on (m)tp_5xx or higher.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_add_scalar](#) for more details.

3.19.49 __bang_subs

```
void __bang_subs(unsigned char *dst,
                 unsigned char *src0,
                 unsigned char *src1,
                 int elem_count)
```

```
void __bang_subs(unsigned short *dst,
                 unsigned short *src0,
                 unsigned short *src1,
                 int elem_count)
```

```
void __bang_subs(unsigned int *dst,
                 unsigned int *src0,
                 unsigned int *src1,
                 int elem_count)
```

```
void __bang_subs(char *dst,
                 char *src0,
                 char *src1,
                 int elem_count)
```

```
void __bang_subs(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)
```

```
void __bang_subs(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)
```

This function performs saturated subtraction operation element-wisely on <src0> and <src1> and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1> and <dst> must point to __nram__ address space;
- <dst>, <src0> and <src1> can be overlapped;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.50 __bang_subs_scalar

```
void __bang_subs_scalar(char *dst,
                        char *src,
                        char value,
                        int elem_count)
```

```
void __bang_subs_scalar(short *dst,
                        short *src,
                        short value,
                        int elem_count)
```

```
void __bang_subs_scalar(int *dst,
                        int *src,
                        int value,
                        int elem_count)
```

This function subtracts <value> from <elem_count> elements of <src> and saves the saturated result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.8.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts

mtp_372;
 • MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 excepts mtp_372.

Example

- None.

3.19.51 __bang_sum

```
half __bang_sum(half *src,
                 int elem_count)
```

```
bfloat16_t __bang_sum(bfloat16_t *src,
                      int elem_count)
```

```
float __bang_sum(float *src,
                  int elem_count)
```

Accumulates all data from source operand <src>.

Parameters

- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source vector.

Return

- The result of accumulation.

Remark

- <src> must point to __nram__ address space;
- <elem_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500 ;
- CNCC Version: cncc --version >= 3.5.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx .

Example

- None.

3.20 Vector Surpass Functions**3.20.1 __bang_cos**

```
void __bang_cos(float *dst,
                float *src,
                int elem_count)
```

This function computes the cosine of each element in <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for

accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source and destination.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- See the example of [__bang_sin](#) for more details.

3.20.2 __bang_div

```
void __bang_div(half *dst,
                 half *src0,
                 half *src1,
                 int elem_count)
```

```
void __bang_div(float *dst,
                 float *src0,
                 float *src1,
                 int elem_count)
```

```
void __bang_div(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)
```

```
void __bang_div(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)
```

```
void __bang_div(unsigned short *dst,
                unsigned short *src0,
                unsigned short *src1,
                int elem_count)
```

```
void __bang_div(unsigned int *dst,
                unsigned int *src0,
                unsigned int *src1,
                int elem_count)
```

```
void __bang_div(half *dst,
                half *src0,
                half *src1,
                int elem_count)
```

```
void __bang_div(float *dst,
                float *src0,
                float *src1,
                int elem_count)
```

```
void __bang_div(short *dst,
                short *src0,
                short *src1,
                int elem_count)
```

```
void __bang_div(int *dst,
                int *src0,
                int *src1,
                int elem_count)
```

```
void __bang_div(unsigned short *dst,
                unsigned short *src0,
                unsigned short *src1,
                int elem_count)
```

```
void __bang_div(unsigned int *dst,
                unsigned int *src0,
                unsigned int *src1,
                int elem_count)
```

This function computes the division of each element in <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Div Operation Function](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.

- [in] `src1`: The address of second source vector, or the value of second source scalar.
- [in] `elem_count`: Number of elements in source and destination.

Return

- void.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` and `<src0>` must point to `__nram__` address space;
- `<src1>` must point to `__nram__` address space if it is a vector;
- The value of `<src1>` cannot be equal to zero;
- `<dst>`, `<src0>` and `<src1>` (vector address) can be overlapped.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322` excepts 372;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30` excepts `mtp_372`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322` excepts `mtp_372`.

Example

```
#include <bang.h>

#define DATA_NUM 128

__mlu_entry__ void kernel(int *output, int *src) {
    __nram__ int _src[DATA_NUM];
    __nram__ int _output[DATA_NUM];
    __memcpy(_src, src, DATA_NUM * sizeof(int), GDRAM2NRAM);
    __bang_div(_output, _src, 10, DATA_NUM);
    __memcpy(output, _output, DATA_NUM * sizeof(int), NRAM2GDRAM);
}
```

3.20.3 `__bang_log`

```
void __bang_log(float *dst,
                float *src,
                int elem_count)
```

This function computes the binary logarithm of each element in `<src>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source and destination.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- See the example of [__bang_sin](#) for more details.

3.20.4 __bang_pow2

```
void __bang_pow2(float *dst,
                 float *src,
                 int elem_count)
```

This function computes the value of 2 to the power of <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Cambricon@155chb

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source and destination.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

- See the example of [__bang_sin](#) for more details.

3.20.5 __bang_recip

```
void __bang_recip(float *dst,
                  float *src,
                  int elem_count)
```

This function computes the reciprocal of each element in <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source and destination.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to `__nram__` address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322 ;`
- CNCC Version: `cncc --version >= 3.0.0 ;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30 ;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322 .`

Example

- See the example of [__bang_sin](#) for more details.

3.20.6 __bang_rem

```
void __bang_rem(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)
```

```
void __bang_rem(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)
```

```
void __bang_rem(unsigned short *dst,
                 unsigned short *src0,
                 unsigned short *src1,
                 int elem_count)
```

```
void __bang_rem(unsigned int *dst,
                unsigned int *src0,
                unsigned int *src1,
                int elem_count)
```

```
void __bang_rem(short *dst,
                 short *src0,
                 short src1,
                 int elem_count)
```

```
void __bang_rem(int *dst,
                 int *src0,
                 int src1,
                 int elem_count)
```

```
void __bang_rem(unsigned short *dst,
                 unsigned short *src0,
                 unsigned short src1,
                 int elem_count)
```

```
void __bang_rem(unsigned int *dst,
                 unsigned int *src0,
                 unsigned int src1,
                 int elem_count)
```

This function computes the remainder of each element in <src> and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector, or the value of second source scalar.
- [in] elem_count: Number of elements in source and destination.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> and <src0> must point to __nram__ address space;
- <src1> must point to __nram__ address space if it is a vector;
- The value of <src1> cannot be equal to zero;
- <dst>, <src0> and <src1> (vector address) can be overlapped.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 excepts 372;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 excepts mtp_372;

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322` excepts `mtp_372`.

Example

```
#include <bang.h>

#define DATA_NUM 128

__mlu_entry__ void kernel(int *output, int *src) {
    __nram__ int _src[DATA_NUM];
    __nram__ int _output[DATA_NUM];
    __memcpy(_src, src, DATA_NUM * sizeof(int), GDRAM2NRAM);
    __bang_rem(_output, _src, 10, DATA_NUM);
    __memcpy(output, _output, DATA_NUM * sizeof(int), NRAM2GDRAM);
}
```

3.20.7 __bang_rsqrt

```
void __bang_rsqrt(float *dst,
                  float *src,
                  int elem_count)
```

This function computes the reciprocal of square root of each element in `<src>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source and destination.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_sin](#) for more details.

3.20.8 __bang_sin

```
void __bang_sin(float *dst,
                float *src,
                int elem_count)
```

This function computes the sine of each element in <src> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: Number of elements in source and destination.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322 ;
- CNCC Version: cncc --version >= 3.0.0 ;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30 ;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322 .

Example

```
#include <bang.h>

#define DATA_NUM 128

__mlu_entry__ void kernel(float *output, float *src) {
    __nram__ float _src[DATA_NUM];
    __nram__ float _output[DATA_NUM];
    __memcpy(_src, src, DATA_NUM * sizeof(float), GDRAM2NRAM);
    __bang_sin(_output, _src, DATA_NUM);
    __memcpy(output, _output, DATA_NUM * sizeof(float), NRAM2GDRAM);
}
```

3.20.9 __bang_sqrt

```
void __bang_sqrt(float *dst,
                 float *src,
                 int elem_count)
```

This function computes the square root of each element in `<src>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Unary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `elem_count`: Number of elements in source and destination.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<dst>` can be overlapped with `<src>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_sin](#) for more details.

3.21 Vector Type Conversion Functions

3.21.1 __bang_bfloat162float

```
void __bang_bfloat162float(float *dst,
                            bfloat16_t *src,
                            int src_count)
```

This function converts type of `<src>` from `bfloat16_t` to `float` element-wisely and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

Return

- `void`.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> cannot be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162float(dst_tmp, src_tmp, SIZE);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

Cambricon@155chb

3.21.2 __bang_bfloat162half

```
void __bang_bfloat162half(half *dst,
                           bfloat16_t *src,
                           int count)
```

This function converts type of <src> from bfloat16_t to half element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;

- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162half(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.3 __bang_bfloat162half_dn

```
void __bang_bfloat162half_dn(half *dst,
                             bfloat16_t *src,
                             int count)
```

This function converts type of <src> from bfloat16_t to half element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
```

```

__bang_bfloat16half_dn(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.4 __bang_bfloat162half_oz

```

void __bang_bfloat162half_oz(half *dst,
                             bfloat16_t *src,
                             int count)

```

This function converts type of <src> from `bfloat16_t` to `half` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162half_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.5 __bang_bfloat162half_rd

```
void __bang_bfloat162half_rd(half *dst,
                             bfloat16_t *src,
                             int count)
```

This function converts type of <src> from bfloat16_t to half element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162half_rd(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.6 __bang_bfloat162half_rm

```
void __bang_bfloat162half_rm(half *dst,
                             bfloat16_t *src,
                             int count)
```

This function converts type of <src> from bfloat16_t to half element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162half_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.7 `__bang_bfloat162half_rn`

```
void __bang_bfloat162half_rn(half *dst,
                             bfloat16_t *src,
                             int count)
```

This function converts type of `<src>` from `bfloat16_t` to `half` element-wisely in round-nearest-even mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162half_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.8 __bang_bfloat162half_tz

```
void __bang_bfloat162half_tz(half *dst,
                             bfloat16_t *src,
                             int count)
```

This function converts type of `<src>` from `bfloat16_t` to `half` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
```

```

__nram__ bfloat16_t src_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
__bang_bfloat162half_tz(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.9 __bang_bfloat162half_up

```

void __bang_bfloat162half_up(half *dst,
                             bfloat16_t *src,
                             int count)

```

This function converts type of <src> from `bfloat16_t` to `half` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, bfloat16_t *src) {
    __nram__ half dst_nram[LEN];
    __nram__ bfloat16_t src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162half_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.10 __bang_bfloat162int16

```
void __bang_bfloat162int16(int16_t *dst,
                            bfloat16_t *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from bfloat16_t to int16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.11 __bang_bfloat162int16_dn

```
void __bang_bfloat162int16_dn(int16_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.12 __bang_bfloat162int16_oz

```
void __bang_bfloat162int16_oz(int16_t *dst,
                                bfloat16_t *src,
                                int count,
                                int fix_position)
```

This function converts type of <src> from bfloat16_t to int16_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.13 __bang_bfloat162int16_rd

```
void __bang_bfloat162int16_rd(int16_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.14 __bang_bfloat162int16_rm

```
void __bang_bfloat162int16_rm(int16_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.15 __bang_bfloat162int16_rn

```
void __bang_bfloat162int16_rn(int16_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int16_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.16 __bang_bfloat162int16_tz

```
void __bang_bfloat162int16_tz(int16_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.17 __bang_bfloat162int16_up

```
void __bang_bfloat162int16_up(int16_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from bfloat16_t to int16_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int16_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.18 __bang_bfloat162int32

```
void __bang_bfloat162int32(int32_t *dst,
                            bfloat16_t *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.19 __bang_bfloat162int32_dn

```
void __bang_bfloat162int32_dn(int32_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int32_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.20 __bang_bfloat162int32_oz

```
void __bang_bfloat162int32_oz(int32_t *dst,
                                bfloat16_t *src,
                                int count,
                                int fix_position)
```

This function converts type of <src> from bfloat16_t to int32_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.21 __bang_bfloat162int32_rd

```
void __bang_bfloat162int32_rd(int32_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int32_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.22 __bang_bfloat162int32_rm

```
void __bang_bfloat162int32_rm(int32_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int32_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.23 __bang_bfloat162int32_rn

```
void __bang_bfloat162int32_rn(int32_t *dst,
                                bfloat16_t *src,
                                int count,
                                int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int32_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.24 __bang_bfloat162int32_tz

```
void __bang_bfloat162int32_tz(int32_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.25 __bang_bfloat162int32_up

```
void __bang_bfloat162int32_up(int32_t *dst,
                               bfloat16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from bfloat16_t to int32_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.26 __bang_bfloat162int4_dn

```
void __bang_bfloat162int4_dn(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from `bfloat16_t` to `int4` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.27 __bang_bfloat162int4_oz

```
void __bang_bfloat162int4_oz(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from `bfloat16_t` to `int4` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_oz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.28 __bang_bfloat162int4_rd

```
void __bang_bfloat162int4_rd(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from `bfloat16_t` to `int4` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_rd(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.29 __bang_bfloat162int4_rm

```
void __bang_bfloat162int4_rm(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from bfloat16_t to int4 element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_rm(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.30 __bang_bfloat162int4_rn

```
void __bang_bfloat162int4_rn(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from `bfloat16_t` to `int4` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_rn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.31 __bang_bfloat162int4_tz

```
void __bang_bfloat162int4_tz(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from `bfloat16_t` to `int4` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_tz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.32 __bang_bfloat162int4_up

```
void __bang_bfloat162int4_up(int4x2_t *dst,
                             bfloat16_t *src,
                             int size,
                             int dst_position)
```

This function converts type of <src> from `bfloat16_t` to `int4` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int4_up(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.33 __bang_bfloat162int8

```
void __bang_bfloat162int8(int8_t *dst,
                           bfloat16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int8_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.34 __bang_bfloat162int8_dn

```
void __bang_bfloat162int8_dn(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from bfloat16_t to int8_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.35 __bang_bfloat162int8_oz

```
void __bang_bfloat162int8_oz(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from bfloat16_t to int8_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.36 __bang_bfloat162int8_rd

```
void __bang_bfloat162int8_rd(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int8_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.37 __bang_bfloat162int8_rm

```
void __bang_bfloat162int8_rm(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int8_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.38 __bang_bfloat162int8_rn

```
void __bang_bfloat162int8_rn(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int8_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.39 __bang_bfloat162int8_tz

```
void __bang_bfloat162int8_tz(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `bfloat16_t` to `int8_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.40 __bang_bfloat162int8_up

```
void __bang_bfloat162int8_up(int8_t *dst,
                             bfloat16_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from bfloat16_t to int8_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162int8_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.41 __bang_bfloat162tf32

```
void __bang_bfloat162tf32(float *dst,
                           bfloat16_t *src,
                           int count)
```

This function converts type of <src> from bfloat16_t to tf32 element-wisely and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162tf32(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.42 __bang_bfloat162uchar

```
void __bang_bfloat162uchar(unsigned char *dst,
                           bfloat16_t *src,
                           int count)
```

This function converts type of <src> from bfloat16_t to unsigned char element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162uchar(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.43 `__bang_bfloat162uchar_dn`

```
void __bang_bfloat162uchar_dn(unsigned char *dst,
                               bfloat16_t *src,
                               int count)
```

This function converts type of <src> from `bfloat16_t` to `unsigned char` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162uchar_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.44 __bang_bfloat162uchar_oz

```
void __bang_bfloat162uchar_oz(unsigned char *dst,
                               bfloat16_t *src,
                               int count)
```

This function converts type of `<src>` from `bfloat16_t` to `unsigned char` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
```

```

__nram__ unsigned char dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
__bang_bfloat162uchar_oz(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}

```

3.21.45 __bang_bfloat162uchar_rd

```

void __bang_bfloat162uchar_rd(unsigned char *dst,
                               bfloat16_t *src,
                               int count)

```

This function converts type of <src> from `bfloat16_t` to `unsigned char` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162uchar_rd(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}

```

3.21.46 __bang_bfloat162uchar_rm

```
void __bang_bfloat162uchar_rm(unsigned char *dst,
                               bfloat16_t *src,
                               int count)
```

This function converts type of <src> from `bfloat16_t` to `unsigned char` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162uchar_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.47 __bang_bfloat162uchar_rn

```
void __bang_bfloat162uchar_rn(unsigned char *dst,
                               bfloat16_t *src,
                               int count)
```

This function converts type of <src> from `bfloat16_t` to `unsigned char` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162uchar_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.48 __bang_bfloat162uchar_tz

```
void __bang_bfloat162uchar_tz(unsigned char *dst,
                               bfloat16_t *src,
                               int count)
```

This function converts type of `<src>` from `bfloat16_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
    __bang_bfloat162uchar_tz(dst_nram, src_nram);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.49 __bang_bfloat162uchar_up

```
void __bang_bfloat162uchar_up(unsigned char *dst,
                               bfloat16_t *src,
                               int count)
```

This function converts type of `<src>` from `bfloat16_t` to `unsigned char` element-wisely in round-up mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, bfloat16_t *src) {
    __nram__ bfloat16_t src_nram[LEN];
```

```

__nram__ unsigned char dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(bfloat16_t), GDRAM2NRAM);
__bang_bfloat162uchar_up(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}

```

3.21.50 __bang_float2bfloat16_dn

```

void __bang_float2bfloat16_dn(bfloat16_t *dst,
                               float *src,
                               int src_count)

```

This function converts type of <src> from `float` to `bfloat16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.1.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_float2bfloat16_rm](#) for more details.

3.21.51 __bang_float2bfloat16_oz

```

void __bang_float2bfloat16_oz(bfloat16_t *dst,
                               float *src,
                               int src_count)

```

This function converts type of <src> from `float` to `bfloat16_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.

- [in] `src_count`: The elements number of conversion.

Return

- void.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.1.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_float2bfloat16_rm](#) for more details.

3.21.52 __bang_float2bfloat16_rd

```
void __bang_float2bfloat16_rd(bfloat16_t *dst,
                               float *src,
                               int src_count)
```

This function converts type of <src> from `float` to `bfloat16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

Return

- void.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.1.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_float2bfloat16_rm](#) for more details.

3.21.53 __bang_float2bfloat16_rm

```
void __bang_float2bfloat16_rm(bfloat16_t *dst,
                             float *src,
                             int src_count)
```

This function converts type of <src> from `float` to `bfloat16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.1.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2bfloat16_rm(dst_tmp, src_tmp, SIZE);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.54 __bang_float2bfloat16_rn

```
void __bang_float2bfloat16_rn(bfloat16_t *dst,
                               float *src,
                               int src_count)
```

This function converts type of <src> from float to bfloat16_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.1.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_float2bfloat16_rm](#) for more details.

3.21.55 __bang_float2bfloat16_tz

```
void __bang_float2bfloat16_tz(bfloat16_t *dst,
                               float *src,
                               int src_count)
```

This function converts type of <src> from float to bfloat16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.1.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of [__bang_float2bfloat16_rm](#) for more details.

3.21.56 __bang_float2bfloat16_up

```
void __bang_float2bfloat16_up(bfloat16_t *dst,
                             float *src,
                             int src_count)
```

This function converts type of `<src>` from `float` to `bfloat16_t` element-wisely in round-up mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

Return

- `void`.

Cambricon@155chb

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src>` can be overlapped with `<dst>`;
- `<src_count>` must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.1.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of [__bang_float2bfloat16_rm](#) for more details.

3.21.57 __bang_float2half_dn

```
void __bang_float2half_dn(half *dst,
                           float *src,
                           int src_count)
```

```
void __bang_float2half_dn(half *dst,
                           float *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of <src> from `float` to `half` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). <src> includes <segnum> + 1 blocks, and each block consists of <src_stride> bytes. <dst> includes <segnum> + 1 blocks, and each block consists of <dst_stride> bytes. In each block of <src>, this function converts first <src_count> elements in round-down mode, and saves the result in blocks in <dst> sequentially. If <src_stride> is zero, this function only converts the first block <segnum> + 1 times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 64 on `(m)tp_2xx`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- <src_stride> must be greater than or equal to <src_count> * `sizeof(half)`, and divisible by `sizeof(float)`;
- $\frac{\text{src_stride}}{\text{sizeof(float)}} \geq \text{src_count}$ if <src_stride> is greater than zero;
- <segnum> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__bang_float2half_tz` for more details.

3.21.58 __bang_float2half_oz

```
void __bang_float2half_oz(half *dst,
                          float *src,
                          int src_count)

void __bang_float2half_oz(half *dst,
                          float *src,
                          int src_count,
                          int dst_stride,
                          int src_stride,
                          int segnum)
```

This function converts type of `<src>` from `float` to `half` element-wisely in round-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-off-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(half)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(float)`;
- $<\text{src_stride}> \div \text{sizeof}(\text{float}) \geq <\text{src_count}>$ if `<src_stride>` is greater than zero;

- <segnum> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of `__bang_float2half_tz` for more details.

3.21.59 `__bang_float2half_rd`

```
void __bang_float2half_rd(half *dst,
                           float *src,
                           int src_count)
```

```
void __bang_float2half_rd(half *dst,
                           float *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of <src> from float to half element-wisely in round-nearest-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). <src> includes <segnum> + 1 blocks, and each block consists of <src_stride> bytes. <dst> includes <segnum> + 1 blocks, and each block consists of <dst_stride> bytes. In each block of <src>, this function converts first <src_count> elements in round-nearest-off-zero mode, and saves the result in blocks in <dst> sequentially. If <src_stride> is zero, this function only converts the first block <segnum> + 1 times.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] dst_stride: The destination stride in bytes.
- [in] src_stride: The source stride in bytes.
- [in] segnum: The number of segments minus one.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 64 on (m)tp_2xx;
- <dst_stride> must be greater than or equal to <src_count> * sizeof(half), and divisible by 64 on (m)tp_2xx, and divisible by sizeof(half) on (m)tp_3xx or higher;
- <src_stride> must be greater than or equal to zero, and divisible by sizeof(float);
- $\frac{\text{src_stride}}{\text{sizeof(float)}} \geq \text{src_count}$ if <src_stride> is greater than zero;
- <segnum> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_float2half_tz](#) for more details.

3.21.60 __bang_float2half_rm

```
void __bang_float2half_rm(half *dst,
                          float *src,
                          int src_count)
```

This function converts type of <src> from `float` to `half` element-wisely in round-math mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src_count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2half_rm(dst_tmp, src_tmp, SIZE);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.61 __bang_float2half_rn

```
void __bang_float2half_rn(half *dst,
                           float *src,
                           int src_count)
```

This function converts type of `<src>` from `float` to `half` element-wisely in round-nearest-even mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of `__bang_float2half_rm` for more details.

3.21.62 __bang_float2half_sr

```
void __bang_float2half_sr(half *dst,
                           float *src,
                           int *srv,
                           int count)
```

Converts source vector with element type `float` to destination vector with element type `half` in round-stochastic mode. The number of elements in the source and destination vectors is `<count>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `srv`: The address of stochastic vector.
- [in] `count`: The elements number of conversion.

Return

- `void`.

Remark

- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`;
- `<src>`, `<srv>` and `<dst>` must point to `__nram__` address space.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

- None.

3.21.63 __bang_float2half_tz

```
void __bang_float2half_tz(half *dst,
                           float *src,
                           int src_count)
```

```
void __bang_float2half_tz(half *dst,
                           float *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of `<src>` from `float` to `half` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information. The

data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(half)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(float)`;
- `<src_stride> / sizeof(float) >= <src_count>` if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20

__mlu_entry__ void kernel(half *dst, float *src, int size) {
    __nram__ float src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ half dst_tmp[DST_STRIDE / sizeof(half) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size * sizeof(float), GDRAM2NRAM);
```

```

__bang_float2half_tz(dst_tmp, src_tmp, LEN, DST_STRIDE, SRC_STRIDE, SEG_NUM);
__memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(half)), NRAM2GDRAM);
}

```

3.21.64 __bang_float2half_up

```
void __bang_float2half_up(half *dst,
                           float *src,
                           int src_count)
```

```
void __bang_float2half_up(half *dst,
                           float *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of `<src>` from `float` to `half` element-wisely in round-up mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-up mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(half)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(float)`;
- $<src_stride> \div sizeof(float) \geq <src_count>$ if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_float2half_tz](#) for more details.

3.21.65 __bang_float2int16_dn

```
void __bang_float2int16_dn(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int16_t element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., <dst> $\times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 64 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define SIZE 128
```

```
#define POS 5

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int16_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.66 __bang_float2int16_oz

```
void __bang_float2int16_oz(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int16_t element-wisely in round-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 64 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_float2int16_dn](#) for more details.

3.21.67 __bang_float2int16_rd

```
void __bang_float2int16_rd(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `float` to `int16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_float2int16_dn](#) for more details.

3.21.68 __bang_float2int16_rm

```
void __bang_float2int16_rm(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `float` to `int16_t` element-wisely in round-math mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.

- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of `__bang_float2int16_dn` for more details.

3.21.69 __bang_float2int16_rn

```
void __bang_float2int16_rn(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

Cambricon@155chb

This function converts type of `<src>` from `float` to `int16_t` element-wisely in round-nearest-even mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;

- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of `__bang_float2int16_dn` for more details.

3.21.70 __bang_float2int16_tz

```
void __bang_float2int16_tz(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int16_t` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on (m)tp_2xx;
- `<src_count>` must be greater than zero and divisible by 64 on (m)tp_2xx;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__bang_float2int16_dn` for more details.

3.21.71 __bang_float2int16_up

```
void __bang_float2int16_up(int16_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int16_t element-wisely in round-up mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 64 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_float2int16_dn](#) for more details.

3.21.72 __bang_float2int32

```
void __bang_float2int32(int32_t *dst,
                        float *src,
                        int src_count,
                        int fix_position)
```

This function converts type of <src> from float to int32_t element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.

- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.73 `__bang_float2int32_dn`

```
void __bang_float2int32_dn(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-down mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.74 __bang_float2int32_oz

```
void __bang_float2int32_oz(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int32_t element-wisely in round-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;

- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_oz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

Cambricon@155chb

3.21.75 __bang_float2int32_rd

```
void __bang_float2int32_rd(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int32_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_rd(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.76 __bang_float2int32_rm

Cambricon@155chb

```
void __bang_float2int32_rm(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-math mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_rm(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.77 __bang_float2int32_rn

Cambricon@155chb

```
void __bang_float2int32_rn(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-nearest-even mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`

- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_rn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.78 __bang_float2int32_tz

```
void __bang_float2int32_tz(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int32_t element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127,127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_tz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.79 __bang_float2int32_up

```
void __bang_float2int32_up(int32_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-up mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
```

```
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int32_up(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.80 __bang_float2int4_dn

```
void __bang_float2int4_dn(int4x2_t *dst,
                           float *src,
                           int size,
                           int dst_position)
```

This function converts type of <src> from float to int4 element-wisely in round-down mode. and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
```

```

__memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
__bang_float2int4_dn(dst_tmp, src_tmp, SIZE, POS);
__memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}

```

3.21.81 __bang_float2int4_oz

```

void __bang_float2int4_oz(int4x2_t *dst,
                          float *src,
                          int size,
                          int dst_position)

```

This function converts type of <src> from float to int4 element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```

#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_oz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}

```

3.21.82 __bang_float2int4_rd

```
void __bang_float2int4_rd(int4x2_t *dst,
                           float *src,
                           int size,
                           int dst_position)
```

This function converts type of <src> from `float` to `int4` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_rd(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.83 __bang_float2int4_rm

```
void __bang_float2int4_rm(int4x2_t *dst,
                           float *src,
                           int size,
                           int dst_position)
```

This function converts type of <src> from `float` to `int4` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_rm(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.84 __bang_float2int4_rn

```
void __bang_float2int4_rn(int4x2_t *dst,
                           float *src,
                           int size,
                           int dst_position)
```

This function converts type of <src> from float to int4 element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_rn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.85 __bang_float2int4_tz

```
void __bang_float2int4_tz(int4x2_t *dst,
                           float *src,
                           int size,
                           int dst_position)
```

This function converts type of <src> from `float` to `int4` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_tz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.86 __bang_float2int4_up

```
void __bang_float2int4_up(int4x2_t *dst,
                           float *src,
                           int size,
                           int dst_position)
```

This function converts type of <src> from float to int4 element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_up(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.87 __bang_float2int8_dn

```
void __bang_float2int8_dn(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `float` to `int8_t` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int8_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.88 __bang_float2int8_oz

```
void __bang_float2int8_oz(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int8_t element-wisely in round-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 128 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_float2int8_dn](#) for more details.

3.21.89 __bang_float2int8_rd

```
void __bang_float2int8_rd(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from float to int8_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.

- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_float2int8_dn` for more details.

3.21.90 __bang_float2int8_rm

```
void __bang_float2int8_rm(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int8_t` element-wisely in round-math mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;

- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of [__bang_float2int8_dn](#) for more details.

3.21.91 __bang_float2int8_rn

```
void __bang_float2int8_rn(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of `<src>` from `float` to `int8_t` element-wisely in round-nearest-even mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

Cambricon@155chb

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of [__bang_float2int8_dn](#) for more details.

3.21.92 __bang_float2int8_tz

```
void __bang_float2int8_tz(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `float` to `int8_t` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_float2int8_dn](#) for more details.

3.21.93 __bang_float2int8_up

```
void __bang_float2int8_up(int8_t *dst,
                           float *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `float` to `int8_t` element-wisely in round-up mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.

- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_float2int8_dn` for more details.

3.21.94 `__bang_float2tf32`

```
void __bang_float2tf32(float *dst,
                      float *src,
                      int count)
```

This function converts type of `<src>` from `float` to `tf32` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.95 __bang_float2tf32_dn

```
void __bang_float2tf32_dn(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to tf32 element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
```

```
}
```

3.21.96 __bang_float2tf32_oz

```
void __bang_float2tf32_oz(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to tf32 element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.97 __bang_float2tf32_rd

```
void __bang_float2tf32_rd(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to tf32 element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_rd(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.98 __bang_float2tf32_rm

```
void __bang_float2tf32_rm(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to tf32 element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.99 __bang_float2tf32_rn

```
void __bang_float2tf32_rn(float *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `float` to `tf32` element-wisely in round-nearest-even mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.100 __bang_float2tf32_sr

```
void __bang_float2tf32_sr(float *dst,
                           float *src,
                           int *srv,
                           int count)
```

Converts source vector with element type `float` to destination vector with element type `tf32` in round-stochastic mode. The number of elements in the source and destination vectors is `<count>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `srv`: The address of stochastic vector.
- [in] `count`: The elements number of conversion.

Return

- `void`.

Remark

- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`;
- `<src>, <srv>` and `<dst>` must point to `__nram__` address space.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

- None.

3.21.101 __bang_float2tf32_tz

```
void __bang_float2tf32_tz(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to tf32 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_tz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.102 __bang_float2tf32_up

```
void __bang_float2tf32_up(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to tf32 element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2tf32_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.103 __bang_float2uchar

```
void __bang_float2uchar(unsigned char *dst,
                        float *src,
                        int count)
```

This function converts type of `<src>` from `float` to `unsigned char` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.104 __bang_float2uchar_dn

```
void __bang_float2uchar_dn(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from float to unsigned char element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.105 __bang_float2uchar_oz

```
void __bang_float2uchar_oz(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from `float` to `unsigned char` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
```

```

__bang_float2uchar_oz(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}

```

3.21.106 __bang_float2uchar_rd

```

void __bang_float2uchar_rd(unsigned char *dst,
                           float *src,
                           int count)

```

This function converts type of <src> from `float` to `unsigned char` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar_rd(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}

```

3.21.107 __bang_float2uchar_rm

```
void __bang_float2uchar_rm(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from `float` to `unsigned char` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.108 __bang_float2uchar_rn

```
void __bang_float2uchar_rn(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from `float` to `unsigned char` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.109 __bang_float2uchar_tz

```
void __bang_float2uchar_tz(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from `float` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar_tz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.110 __bang_float2uchar_up

```
void __bang_float2uchar_up(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from `float` to `unsigned char` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_float2uchar_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.111 __bang_half2bfloat16

```
void __bang_half2bfloat16(bfloat16_t *dst,
                          half *src,
                          int count)
```

This function converts type of <src> from half to bfloat16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.112 __bang_half2bfloat16_dn

```
void __bang_half2bfloat16_dn(bfloat16_t *dst,
                           half *src,
                           int count)
```

This function converts type of <src> from half to bfloat16_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.113 `__bang_half2bfloat16_oz`

```
void __bang_half2bfloat16_oz(bfloat16_t *dst,
                             half *src,
                             int count)
```

This function converts type of `<src>` from `half` to `bfloat16_t` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.114 __bang_half2bfloat16_rd

```
void __bang_half2bfloat16_rd(bfloat16_t *dst,
                             half *src,
                             int count)
```

This function converts type of `<src>` from `half` to `bfloat16_t` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
```

```

__nram__ half src_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
__bang_half2bfloat16_rd(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}

```

3.21.115 __bang_half2bfloat16_rm

```

void __bang_half2bfloat16_rm(bfloat16_t *dst,
                             half *src,
                             int count)

```

This function converts type of <src> from `half` to `bfloat16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}

```

3.21.116 __bang_half2bfloat16_rn

```
void __bang_half2bfloat16_rn(bfloat16_t *dst,
                             half *src,
                             int count)
```

This function converts type of <src> from half to bfloat16_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.117 __bang_half2bfloat16_tz

```
void __bang_half2bfloat16_tz(bfloat16_t *dst,
                            half *src,
                            int count)
```

This function converts type of <src> from half to bfloat16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16_tz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.118 __bang_half2bfloat16_up

```
void __bang_half2bfloat16_up(bfloat16_t *dst,
                             half *src,
                             int count)
```

This function converts type of `<src>` from `half` to `bfloat16_t` element-wisely in round-up mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, half *src) {
    __nram__ bfloat16_t dst_nram[LEN];
    __nram__ half src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2bfloat16_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.119 __bang_half2float

```
void __bang_half2float(float *dst,
                      half *src,
                      int src_count)
```

```
void __bang_half2float(float *dst,
                      half *src,
                      int src_count,
                      int dst_stride,
                      int src_stride,
                      int segnum)
```

This function converts type of `<src>` from `half` to `float` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 64 on (m)tp_2xx;
- <dst_stride> must be greater than or equal to <src_count> * sizeof(float), and divisible by 64 on (m)tp_2xx, and divisible by sizeof(float) on (m)tp_3xx or higher;
- <src_stride> must be greater than or equal to zero, and divisible by sizeof(half);
- $\frac{\text{src_stride}}{\text{sizeof(half)}} \geq \text{src_count}$ if <src_stride> is greater than zero;
- <segnum> must be greater than or equal to zero;
- <src> cannot be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>
Cambricon@155chb
#define SRC_STRIDE 160
#define DST_STRIDE 512
#define LEN 128
#define SEG_NUM 20

__mlu_entry__ void kernel(float *dst, half *src, int size) {
    __nram__ half src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ float dst_tmp[DST_STRIDE / sizeof(float) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size * sizeof(half), GDRAM2NRAM);
    __bang_half2float(dst_tmp, src_tmp, LEN, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(float)), NRAM2GDRAM);
}
```

3.21.120 __bang_half2int16_dn

```
void __bang_half2int16_dn(int16_t *dst,
                          half *src,
                          int src_count,
                          int fix_position)
```

This function converts type of <src> from half to int16_t element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 64 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>
Cambricon@155chb
#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int16_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int16_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.121 __bang_half2int16_oz

```
void __bang_half2int16_oz(int16_t *dst,
                           half *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from half to int16_t element-wisely in round-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `--BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_half2int16_dn` for more details.

3.21.122 __bang_half2int16_rd

```
void __bang_half2int16_rd(int16_t *dst,
                          half *src,
                          int src_count,
                          int fix_position)
```

This function converts type of `<src>` from `half` to `int16_t` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_half2int16_dn](#) for more details.

3.21.123 __bang_half2int16_rm

```
void __bang_half2int16_rm(int16_t *dst,
                          half *src,
                          int src_count,
                          int fix_position)
```

This function converts type of <src> from `half` to `int16_t` element-wisely in round-math mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_half2int16_dn](#) for more details.

3.21.124 __bang_half2int16_rn

```
void __bang_half2int16_rn(int16_t *dst,
                           half *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int16_t` element-wisely in round-nearest-even mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_half2int16_dn](#) for more details.

3.21.125 __bang_half2int16_tz

```
void __bang_half2int16_tz(int16_t *dst,
                           half *src,
                           int src_count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.

- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<src>` can be overlapped with `<dst>`;
- `<fix_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_half2int16_dn](#) for more details.

3.21.126 __bang_half2int16_up

```
void __bang_half2int16_up(int16_t *dst,
                           half *src,
                           int src_count,
                           int fix_position)
```

Cambricon@155chb

This function converts type of `<src>` from `half` to `int16_t` element-wisely in round-up mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;

- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of [__bang_half2int16_dn](#) for more details.

3.21.127 __bang_half2int32

```
void __bang_half2int32(int32_t *dst,
                      half *src,
                      int count,
                      int fix_position)
```

This function converts type of `<src>` from `half` to `int32_t` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` cannot be overlapped with `<dst>`;
- `<fix_position>` must be in the range $[-127, 127]$.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
```

```
}
```

3.21.128 __bang_half2int32_dn

```
void __bang_half2int32_dn(int32_t *dst,
                           half *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int32_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.129 __bang_half2int32_oz

```
void __bang_half2int32_oz(int32_t *dst,
                           half *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int32_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.130 __bang_half2int32_rd

```
void __bang_half2int32_rd(int32_t *dst,
                           half *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int32_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.131 __bang_half2int32_rm

```
void __bang_half2int32_rm(int32_t *dst,
                          half *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from half to int32_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.132 __bang_half2int32_rn

```
void __bang_half2int32_rn(int32_t *dst,
                           half *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int32_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.133 __bang_half2int32_tz

```
void __bang_half2int32_tz(int32_t *dst,
                           half *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `half` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.134 __bang_half2int32_up

```
void __bang_half2int32_up(int32_t *dst,
                          half *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `half` to `int32_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2int32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.135 __bang_half2int4_dn

```
void __bang_half2int4_dn(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from `half` to `int4` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int4_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.136 __bang_half2int4_oz

```
void __bang_half2int4_oz(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from `half` to `int4` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int4_oz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.137 __bang_half2int4_rd

```
void __bang_half2int4_rd(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from `half` to `int4` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int4_tz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.138 __bang_half2int4_rm

```
void __bang_half2int4_rm(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from `half` to `int4` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int4_rm(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.139 __bang_half2int4_rn

```
void __bang_half2int4_rn(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from `half` to `int4` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int4_rn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.140 __bang_half2int4_tz

```
void __bang_half2int4_tz(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from half to int4 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int4_tz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.141 __bang_half2int4_up

```
void __bang_half2int4_up(int4x2_t *dst,
                         half *src,
                         int size,
                         int dst_position)
```

This function converts type of <src> from `half` to `int4` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2int4_up(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.142 __bang_half2int8_dn

```
void __bang_half2int8_dn(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)
```

```
void __bang_half2int8_dn(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position,
                         int dst_stride,
                         int src_stride,
                         int segnum)
```

This function converts type of <src> from `half` to `int8_t` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). <src> includes <segnum> + 1 blocks, and each block consists of <src_stride> bytes. <dst> includes <segnum> + 1 blocks, and each block consists of <dst_stride> bytes. In each block of <src>, this function converts first <src_count> elements according to <fix_position> in round-down mode, and saves the result in blocks in <dst> sequentially. If <src_stride> is zero, this function only converts the first block <segnum> + 1 times.

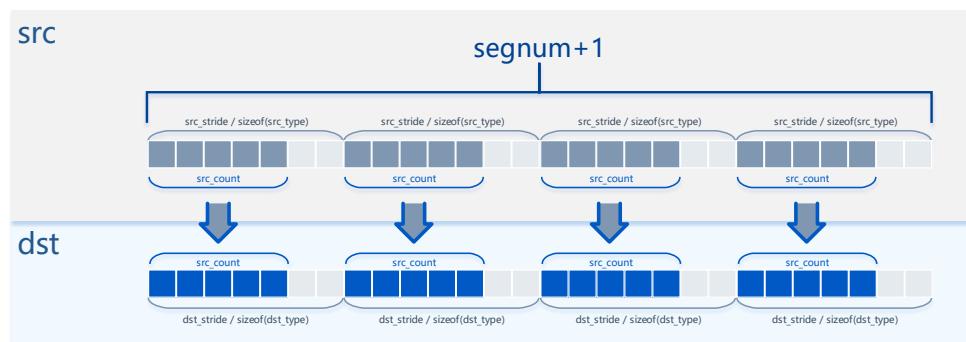


Fig. 3.16: The Process of Conversion With Stride

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<fix_position>}$.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 128 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <dst_stride> must be greater than or equal to <src_count> * sizeof(int8_t), and divisible by 64 on (m)tp_2xx, and divisible by sizeof(int8_t) on (m)tp_3xx or higher;
- <src_stride> must be greater than or equal to zero, and divisible by sizeof(half);
- <src_stride> \div sizeof(half) \geq <src_count> if <src_stride> is greater than zero;
- <segnum> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ \geq 200;
- CNCC Version: cncc --version \geq 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch \geq compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch \geq (m)tp_2xx.

Example

```
#include <bang.h>
Cambricon@155chb

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20
#define POS 5

__mlu_entry__ void kernel(int8_t *dst, half *src, int size) {
    __nram__ half src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ int8_t dst_tmp[DST_STRIDE / sizeof(int8_t) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size, GDRAM2NRAM);
    __bang_half2int8_dn(dst_tmp, src_tmp, LEN, POS, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(half)), NRAM2GDRAM);
}
```

3.21.143 __bang_half2int8_oz

```
void __bang_half2int8_oz(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)

void __bang_half2int8_oz(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position,
                         int dst_stride,
                         int src_stride,
                         int segnum)
```

This function converts type of `<src>` from `half` to `int8_t` element-wisely in round-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements according to `<fix_position>` in round-off-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<fix_position>}$.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(int8_t)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(int8_t)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(half)`;
- $<src_stride> \div sizeof(half) \geq <src_count>$ if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of `__bang_half2int8_dn` for more details.

3.21.144 `__bang_half2int8_rd`

```
void __bang_half2int8_rd(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)

void __bang_half2int8_rd(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position,
                         int dst_stride,
                         int src_stride,
                         int segnum)
```

Cambricon@155chb

This function converts type of `<src>` from `half` to `int8_t` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements according to `<fix_position>` in round-nearest-off-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 128 on (m)tp_2xx;
- <fix_position> must be in the range [-127,127];
- <dst_stride> must be greater than or equal to <src_count> * sizeof(int8_t), and divisible by 64 on (m)tp_2xx, and divisible by sizeof(int8_t) on (m)tp_3xx or higher;
- <src_stride> must be greater than or equal to zero, and divisible by sizeof(half);
- $\frac{\text{src_stride}}{\text{sizeof(half)}} \geq \text{src_count}$ if <src_stride> is greater than zero;
- <segnorm> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_half2int8_dn](#) for more details.

3.21.145 __bang_half2int8_rm

```
void __bang_half2int8_rm(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)
```

This function converts type of <src> from half to int8_t element-wisely in round-math mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $\text{dst} \times 2^{\text{fix_position}}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <fix_position> must be in the range [-127,127];
- <src_count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_half2int8_dn](#) for more details.

3.21.146 __bang_half2int8_rn

```
void __bang_half2int8_rn(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)
```

This function converts type of `<src>` from `half` to `int8_t` element-wisely in round-nearest-even mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<fix_position>` must be in the range [-127, 127];
- `<src_count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_half2int8_dn](#) for more details.

3.21.147 __bang_half2int8_tz

```
void __bang_half2int8_tz(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)

void __bang_half2int8_tz(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position,
                         int dst_stride,
                         int src_stride,
                         int segnum)
```

This function converts type of `<src>` from `half` to `int8_t` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements according to `<fix_position>` in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<fix_position>}$.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(int8_t)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(int8_t)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(half)`;
- $<src_stride> \div sizeof(half) \geq <src_count>$ if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__bang_half2int8_dn` for more details.

3.21.148 __bang_half2int8_up

```
void __bang_half2int8_up(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position)

void __bang_half2int8_up(int8_t *dst,
                         half *src,
                         int src_count,
                         int fix_position,
                         int dst_stride,
                         int src_stride,
                         int segnum)
```

Cambricon@155chb

This function converts type of `<src>` from `half` to `int8_t` element-wisely in round-up mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements according to `<fix_position>` in round-up mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_count> must be greater than zero and divisible by 128 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <dst_stride> must be greater than or equal to <src_count> * sizeof(int8_t), and divisible by 64 on (m)tp_2xx, and divisible by sizeof(int8_t) on (m)tp_3xx or higher;
- <src_stride> must be greater than or equal to zero, and divisible by sizeof(half);
- $\frac{\text{src_stride}}{\text{sizeof(half)}} \geq \text{src_count}$ if <src_stride> is greater than zero;
- <segnr> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_half2int8_dn](#) for more details.

3.21.149 __bang_half2short_dn

```
void __bang_half2short_dn(short *dst,
                           half *src,
                           int src_count)
```

```
void __bang_half2short_dn(short *dst,
                           half *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of <src> from `half` to `short` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). <src> includes <segnr> + 1 blocks, and each block consists of <src_stride> bytes. <dst> includes <segnr> + 1 blocks, and each block consists of <dst_stride> bytes. In each block of <src>, this function converts first <src_count> elements in round-down mode, and saves the result in blocks in <dst> sequentially. If <src_stride> is zero, this function only converts the first block <segnr> + 1 times.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnun`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(short)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(half)`;
- $\frac{<\text{src_stride}>}{\text{sizeof(half)}} \geq <\text{src_count}>$ if `<src_stride>` is greater than zero;
- `<segnun>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20

__mlu_entry__ void kernel(short *dst, half *src, int size) {
    __nram__ half src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ short dst_tmp[DST_STRIDE / sizeof(short) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size * sizeof(half), GDRAM2NRAM);
    __bang_half2short_dn(dst_tmp, src_tmp, LEN, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(short)), NRAM2GDRAM);
}
```

3.21.150 __bang_half2short_oz

```
void __bang_half2short_oz(short *dst,
                           half *src,
                           int src_count)
```

```
void __bang_half2short_oz(short *dst,
                           half *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of `<src>` from `half` to `short` element-wisely in round-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-off-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to `<src_count> * sizeof(short)`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride> / sizeof(half) ≥ <src_count>` if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.12.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- See the example of `__bang_half2short_dn` for more details.

3.21.151 __bang_half2short_rd

```
void __bang_half2short_rd(short *dst,
                           half *src,
                           int src_count)

void __bang_half2short_rd(short *dst,
                           half *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of `<src>` from `half` to `short` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-nearest-off-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(short)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(half)`;
- $\frac{<src_stride>}{\text{sizeof}(half)} \geq <\text{src_count}>$ if `<src_stride>` is greater than zero;

- <segnum> must be greater than or equal to zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_half2short_dn](#) for more details.

3.21.152 __bang_half2short_rm

```
void __bang_half2short_rm(short *dst,
                           half *src,
                           int src_count)
```

This function converts type of <src> from half to short element-wisely in round-math mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Cambricon@155chb

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src_count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(short *dst, half *src) {
```

```

__nram__ half src_tmp[SIZE];
__nram__ short dst_tmp[SIZE];
__memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
__bang_half2short_rm(dst_tmp, src_tmp, SIZE);
__memcpy(dst, dst_tmp, SIZE * sizeof(short), NRAM2GDRAM);
}

```

3.21.153 __bang_half2short_rn

```

void __bang_half2short_rn(short *dst,
                          half *src,
                          int src_count)

```

This function converts type of <src> from half to short element-wisely in round-nearest-even mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.

Return

- void.

Remark

Cambricon@155chb

- <src> and <dst> must point to __nram__ address space;
- <src_count> must be greater than zero;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_half2short_rm](#) for more details.

3.21.154 __bang_half2short_tz

```

void __bang_half2short_tz(short *dst,
                          half *src,
                          int src_count)

```

```
void __bang_half2short_tz(short *dst,
                          half *src,
                          int src_count,
                          int dst_stride,
                          int src_stride,
                          int segnum)
```

This function converts type of `<src>` from `half` to `short` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(short)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(half)`;
- $<src_stride> \div sizeof(half) \geq <src_count>$ if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- See the example of [__bang_half2short_dn](#) for more details.

3.21.155 __bang_half2short_up

```
void __bang_half2short_up(short *dst,
                           half *src,
                           int src_count)
```

```
void __bang_half2short_up(short *dst,
                           half *src,
                           int src_count,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

This function converts type of `<src>` from `half` to `short` element-wisely in round-up mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-up mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to `<src_count> * sizeof(short)`, and divisible by `sizeof(short)` on `(m)tp_3xx` or higher;
- `<src_stride> / sizeof(half) ≥ <src_count>` if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_half2short_dn](#) for more details.

3.21.156 __bang_half2tf32

```
void __bang_half2tf32(float *dst,
                      half *src,
                      int count)
```

This function converts type of <src> from half to tf32 element-wisely and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, half *src) {
    __nram__ half src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(half), GDRAM2NRAM);
    __bang_half2tf32(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.157 __bang_half2uchar_dn

```
void __bang_half2uchar_dn(unsigned char *dst,
                           half *src,
                           int src_count)
```

```
void __bang_half2uchar_dn(unsigned char *dst,
                           half *src,
                           half *src_addition,
                           int src_count)
```

This function converts type of <src> from `half` to `unsigned char` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_addition`: The address of additional vector for source.
- [in] `src_count`: The number of elements.

Return

- `void`.

Remark

- <src>, <dst> and <src_addition> must point to `__nram__` address space;
- The size of <src_addition> vector is identical to the size of <src> and <dst>;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- The difference between two version is that the version without <src_addition> can convert data in the range of `[0, 127]`, the version with <src_addition> can convert data in the range of `[0, 255]`;
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(unsigned char *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ unsigned char dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
```

```

__bang_half2uchar_dn(dst_tmp, src_tmp, SIZE);
__memcpy(dst, dst_tmp, SIZE * sizeof(unsigned char), NRAM2GDRAM);
}

```

3.21.158 __bang_int162bfloating16

```

void __bang_int162bfloating16(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)

```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```

#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloating16(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}

```

3.21.159 __bang_int162bfloat16_dn

```
void __bang_int162bfloat16_dn(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.160 __bang_int162bfloat16_oz

```
void __bang_int162bfloat16_oz(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 592;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq `mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.161 __bang_int162bfloat16_rd

```
void __bang_int162bfloat16_rd(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.162 __bang_int162bfloat16_rm

```
void __bang_int162bfloat16_rm(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.163 __bang_int162bfloat16_rn

```
void __bang_int162bfloat16_rn(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.164 __bang_int162bfloat16_tz

```
void __bang_int162bfloat16_tz(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.165 __bang_int162bfloat16_up

```
void __bang_int162bfloat16_up(bfloat16_t *dst,
                               int16_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int16_t` to `bfloat16_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162bfloat16_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.166 __bang_int162float

```
void __bang_int162float(float *dst,
                        int16_t *src,
                        int src_count,
                        int fix_position)
```

This function converts type of <src> from `int16_t` to `float` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- <fix_position> must be in the range [-127, 127];
- <src> cannot be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_int162float(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.167 __bang_int162half

```
void __bang_int162half(half *dst,
                      int16_t *src,
                      int src_count,
                      int fix_position)
```

This function converts type of <src> from `int16_t` to `half` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 64 on `(m)tp_2xx`;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(half *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162half(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.168 __bang_int162int32

```
void __bang_int162int32(int32_t *dst,
                        int16_t *src,
                        int src_count,
                        int dst_position,
                        int src_position)
```

This function converts type of <src> from `int16_t` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127, 127];
- <src> cannot be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc -bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int32_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int32(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.169 __bang_int162int4_dn

```
void __bang_int162int4_dn(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of `<src>` from `int16_t` to `int4` element-wisely in round-down mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_dn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.170 __bang_int162int4_oz

```
void __bang_int162int4_oz(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of `<src>` from `int16_t` to `int4` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_tt src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.171 __bang_int162int4_rd

```
void __bang_int162int4_rd(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int16_t` to `int4` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_rd(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.172 __bang_int162int4_rm

```
void __bang_int162int4_rm(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int16_t` to `int4` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.173 __bang_int162int4_rn

```
void __bang_int162int4_rn(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int16_t` to `int4` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_rn(dst_tmp, src_tmp, SIZE);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.174 __bang_int162int4_tz

```
void __bang_int162int4_tz(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int16_t to int4 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_tz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.175 __bang_int162int4_up

```
void __bang_int162int4_up(int4x2_t *dst,
                           int16_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int16_t` to `int4` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int4_up(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.176 __bang_int162int8

```
void __bang_int162int8(int8_t *dst,
                      int16_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of <src> from `int16_t` to `int8_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int8_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162int8(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.177 __bang_int162tf32

```
void __bang_int162tf32(float *dst,
                        int16_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from int16_t to tf32 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.178 __bang_int162tf32_dn

```
void __bang_int162tf32_dn(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `tf32` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.179 __bang_int162tf32_oz

```
void __bang_int162tf32_oz(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from int16_t to tf32 element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.180 __bang_int162tf32_rd

```
void __bang_int162tf32_rd(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from int16_t to tf32 element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.181 __bang_int162tf32_rm

```
void __bang_int162tf32_rm(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `tf32` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.182 __bang_int162tf32_rn

```
void __bang_int162tf32_rn(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `tf32` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.183 __bang_int162tf32_tz

```
void __bang_int162tf32_tz(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from int16_t to tf32 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.184 __bang_int162tf32_up

```
void __bang_int162tf32_up(float *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from int16_t to tf32 element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162tf32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.185 __bang_int162uchar

```
void __bang_int162uchar(unsigned char *dst,
                        int16_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.186 __bang_int162uchar_dn

```
void __bang_int162uchar_dn(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.187 __bang_int162uchar_oz

```
void __bang_int162uchar_oz(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.188 __bang_int162uchar_rd

```
void __bang_int162uchar_rd(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.189 __bang_int162uchar_rm

```
void __bang_int162uchar_rm(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.190 __bang_int162uchar_rn

```
void __bang_int162uchar_rn(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.191 __bang_int162uchar_tz

```
void __bang_int162uchar_tz(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.192 __bang_int162uchar_up

```
void __bang_int162uchar_up(unsigned char *dst,
                           int16_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int16_t` to `unsigned char` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int16_t *src) {
    __nram__ int16_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int162uchar_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.193 __bang_int322bfloat16

```
void __bang_int322bfloat16(bfloat16_t *dst,
                            int32_t *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.194 __bang_int322bfloat16_dn

```
void __bang_int322bfloat16_dn(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.195 __bang_int322bfloat16_oz

```
void __bang_int322bfloat16_oz(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.196 __bang_int322bfloat16_rd

```
void __bang_int322bfloat16_rd(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.197 __bang_int322bfloat16_rm

```
void __bang_int322bfloat16_rm(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.198 __bang_int322bfloat16_rn

```
void __bang_int322bfloat16_rn(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.199 __bang_int322bfloat16_tz

```
void __bang_int322bfloat16_tz(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int16_t), GDRAM2NRAM);
    __bang_int322bfloat16_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.200 __bang_int322bfloat16_up

```
void __bang_int322bfloat16_up(bfloat16_t *dst,
                               int32_t *src,
                               int count,
                               int fix_position)
```

This function converts type of <src> from `int32_t` to `bfloat16_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322bfloat16_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.201 __bang_int322float

```
void __bang_int322float(float *dst,
                        int32_t *src,
                        int src_count,
                        int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.202 __bang_int322float_dn

```
void __bang_int322float_dn(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<\text{src}> \times 2^{<\text{fix_position}>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.203 __bang_int322float_oz

```
void __bang_int322float_oz(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_oz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.204 __bang_int322float_rd

```
void __bang_int322float_rd(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-nearest-off-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_rd(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.205 __bang_int322float_rm

```
void __bang_int322float_rm(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-math mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_rm(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.206 __bang_int322float_rn

```
void __bang_int322float_rn(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-nearest-even mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_rn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.207 __bang_int322float_tz

```
void __bang_int322float_tz(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_tz(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.208 __bang_int322float_up

```
void __bang_int322float_up(float *dst,
                            int32_t *src,
                            int src_count,
                            int fix_position)
```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-up mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>.

Instruction Pipeline

Cambricon@155chb

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322float_up(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.209 __bang_int322half

```
void __bang_int322half(half *dst,
                      int32_t *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.210 __bang_int322half_dn

```
void __bang_int322half_dn(half *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.211 __bang_int322half_oz

```
void __bang_int322half_oz(half *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.212 __bang_int322half_rd

```
void __bang_int322half_rd(half *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.213 __bang_int322half_rm

```
void __bang_int322half_rm(half *dst,
                          int32_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.214 __bang_int322half_rn

```
void __bang_int322half_rn(half *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.215 __bang_int322half_tz

```
void __bang_int322half_tz(half *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.216 __bang_int322half_up

```
void __bang_int322half_up(half *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `half` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(half *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322half_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.217 __bang_int322int16

```
void __bang_int322int16(int16_t *dst,
                        int32_t *src,
                        int src_count,
                        int dst_position,
                        int src_position)
```

This function converts type of <src> from `int32_t` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int16_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int16(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.218 __bang_int322int4_dn

```
void __bang_int322int4_dn(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of `<src>` from `int32_t` to `int4` element-wisely in round-down mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_dn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.219 __bang_int322int4_oz

```
void __bang_int322int4_oz(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of `<src>` from `int32_t` to `int4` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `_nram_` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.220 __bang_int322int4_rd

```
void __bang_int322int4_rd(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int32_t` to `int4` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_rd(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.221 __bang_int322int4_rm

```
void __bang_int322int4_rm(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int32_t` to `int4` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.222 __bang_int322int4_rn

```
void __bang_int322int4_rn(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int32_t` to `int4` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_rn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.223 __bang_int322int4_tz

```
void __bang_int322int4_tz(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int32_t to int4 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_tz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.224 __bang_int322int4_up

```
void __bang_int322int4_up(int4x2_t *dst,
                           int32_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int32_t` to `int4` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322int4_up(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.225 __bang_int32int8

```
void __bang_int32int8(int8_t *dst,
                      int32_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of <src> from `int32_t` to `int8_t` element-wisely round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int8_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
    __bang_int32int8(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.226 __bang_int322tf32

```
void __bang_int322tf32(float *dst,
                        int32_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322tf32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.227 __bang_int322tf32_dn

```
void __bang_int322tf32_dn(float *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322tf32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.228 __bang_int322tf32_oz

```
void __bang_int322tf32_oz(float *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322tf32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.229 __bang_int322tf32_rd

```
void __bang_int322tf32_rd(float *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322tf32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.230 __bang_int32tf32_rm

```
void __bang_int32tf32_rm(float *dst,
                          int32_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int32tf32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.231 __bang_int322tf32_rn

```
void __bang_int322tf32_rn(float *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq `(m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322tf32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.232 __bang_int322tf32_tz

```
void __bang_int322tf32_tz(float *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `tf32` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322tf32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.233 __bang_int32tf32_up

```
void __bang_int32tf32_up(float *dst,
                          int32_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from int32_t to tf32 element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int32tf32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.234 __bang_int322uchar

```
void __bang_int322uchar(unsigned char *dst,
                        int32_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{<fix_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx.`

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.235 __bang_int322uchar_dn

```
void __bang_int322uchar_dn(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.236 __bang_int322uchar_oz

```
void __bang_int322uchar_oz(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.237 __bang_int322uchar_rd

```
void __bang_int322uchar_rd(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.238 __bang_int322uchar_rm

```
void __bang_int322uchar_rm(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.239 __bang_int322uchar_rn

```
void __bang_int322uchar_rn(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.240 __bang_int322uchar_tz

```
void __bang_int322uchar_tz(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.241 __bang_int322uchar_up

```
void __bang_int322uchar_up(unsigned char *dst,
                           int32_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int32_t` to `unsigned char` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int32_t *src) {
    __nram__ int32_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int32_t), GDRAM2NRAM);
    __bang_int322uchar_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.242 __bang_int42bfloat16_dn

```
void __bang_int42bfloat16_dn(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from int4 to bfloat16_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_dn(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.243 __bang_int42bfloat16_oz

```
void __bang_int42bfloat16_oz(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from `int4` to `bfloat16_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_oz(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.244 __bang_int42bfloat16_rd

```
void __bang_int42bfloat16_rd(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from `int4` to `bfloat16_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_rd(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.245 __bang_int42bfloat16_rm

```
void __bang_int42bfloat16_rm(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from int4 to bfloat16_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_rm(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.246 __bang_int42bfloat16_rn

```
void __bang_int42bfloat16_rn(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from `int4` to `bfloat16_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements Cambricon@155chb

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_rn(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.247 __bang_int42bfloat16_tz

```
void __bang_int42bfloat16_tz(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from `int4` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_tz(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.248 __bang_int42bfloat16_up

```
void __bang_int42bfloat16_up(bfloat16_t *dst,
                             int4x2_t *src,
                             int size,
                             int src_position)
```

This function converts type of <src> from `int4` to `bfloat16_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(bfloat16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ bfloat16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42bfloat16_up(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.249 __bang_int42float_dn

```
void __bang_int42float_dn(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_dn(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.250 __bang_int42float_oz

```
void __bang_int42float_oz(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{src_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_oz(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.251 __bang_int42float_rd

```
void __bang_int42float_rd(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_rd(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.252 __bang_int42float_rm

```
void __bang_int42float_rm(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{src_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_rm(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.253 __bang_int42float_rn

```
void __bang_int42float_rn(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_rn(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.254 __bang_int42float_tz

```
void __bang_int42float_tz(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_tz(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.255 __bang_int42float_up

```
void __bang_int42float_up(float *dst,
                           int4x2_t *src,
                           int size,
                           int src_position)
```

This function converts type of <src> from `int4` to `float` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42float_up(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.256 __bang_int42half_dn

```
void __bang_int42half_dn(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from `int4` to `half` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_dn(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.257 __bang_int42half_oz

```
void __bang_int42half_oz(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from `int4` to `half` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{src_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_oz(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.258 __bang_int42half_rd

```
void __bang_int42half_rd(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from `int4` to `half` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_rd(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.259 __bang_int42half_rm

```
void __bang_int42half_rm(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from `int4` to `half` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{src_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_rm(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.260 __bang_int42half_rn

```
void __bang_int42half_rn(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from int4 to half element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements Cambricon@155chb

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_rn(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.261 __bang_int42half_tz

```
void __bang_int42half_tz(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from `int4` to `half` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_tz(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.262 __bang_int42half_up

```
void __bang_int42half_up(half *dst,
                         int4x2_t *src,
                         int size,
                         int src_position)
```

This function converts type of <src> from `int4` to `half` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{src_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src> cannot be overlapped with <dst>;
- <size> must be greater than zero and divisible by 2;
- <src_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(half *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ half dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42half_up(dst_tmp, src_tmp, SIZE, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(half), NRAM2GDRAM);
}
```

3.21.263 __bang_int42int16_dn

```
void __bang_int42int16_dn(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int4` to `int16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_dn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.264 __bang_int42int16_oz

```
void __bang_int42int16_oz(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int16_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.265 __bang_int42int16_rd

```
void __bang_int42int16_rd(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int16_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_rd(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.266 __bang_int42int16_rm

```
void __bang_int42int16_rm(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int16_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.267 __bang_int42int16_rn

```
void __bang_int42int16_rn(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int16_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_rn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.268 __bang_int42int16_tz

```
void __bang_int42int16_tz(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_tz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.269 __bang_int42int16_up

```
void __bang_int42int16_up(int16_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int16_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int16_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int16_up(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.270 __bang_int42int32_dn

```
void __bang_int42int32_dn(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from `int4` to `int32_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_dn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.271 __bang_int42int32_oz

```
void __bang_int42int32_oz(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int32_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.272 __bang_int42int32_rd

```
void __bang_int42int32_rd(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int32_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127];

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_rd(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.273 __bang_int42int32_rm

```
void __bang_int42int32_rm(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int32_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.274 __bang_int42int32_rn

```
void __bang_int42int32_rn(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int32_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_rn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.275 __bang_int42int32_tz

```
void __bang_int42int32_tz(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int32_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_tz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.276 __bang_int42int32_up

```
void __bang_int42int32_up(int32_t *dst,
                           int4x2_t *src,
                           int size,
                           int dst_position,
                           int src_position)
```

This function converts type of <src> from int4 to int32_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int32_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int32_up(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.277 __bang_int42int8_dn

```
void __bang_int42int8_dn(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.278 __bang_int42int8_oz

```
void __bang_int42int8_oz(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.279 __bang_int42int8_rd

```
void __bang_int42int8_rd(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_rd(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.280 __bang_int42int8_rm

```
void __bang_int42int8_rm(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.281 __bang_int42int8_rn

```
void __bang_int42int8_rn(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.282 __bang_int42int8_tz

```
void __bang_int42int8_tz(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_tz(dst_tmp, src_tmp, SIZE, DSRPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.283 __bang_int42int8_up

```
void __bang_int42int8_up(int8_t *dst,
                         int4x2_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from int4 to int8_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] size: The elements number of conversion.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <size> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.7.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int8_t *dst, int4x2_t *src) {
    __nram__ int4x2_t src_tmp[SIZE/2];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int4x2_t) / 2, GDRAM2NRAM);
    __bang_int42int8_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.284 __bang_int42tf32

```
void __bang_int42tf32(float *dst,
                      int4x2_t *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from int4x2_t to tf32 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.285 __bang_int42tf32_dn

```
void __bang_int42tf32_dn(float *dst,
                          int4x2_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int4x2_t` to `tf32` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.286 __bang_int42tf32_oz

```
void __bang_int42tf32_oz(float *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from int4x2_t to tf32 element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.287 __bang_int42tf32_rd

```
void __bang_int42tf32_rd(float *dst,
                          int4x2_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int4x2_t` to `tf32` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.288 __bang_int42tf32_rm

```
void __bang_int42tf32_rm(float *dst,
                          int4x2_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int4x2_t` to `tf32` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.289 __bang_int42tf32_rn

```
void __bang_int42tf32_rn(float *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `tf32` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, char *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.290 __bang_int42tf32_tz

```
void __bang_int42tf32_tz(float *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from int4x2_t to tf32 element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.291 __bang_int42tf32_up

```
void __bang_int42tf32_up(float *dst,
                          int4x2_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int4x2_t` to `tf32` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42tf32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.292 __bang_int42uchar

```
void __bang_int42uchar(unsigned char *dst,
                      int4x2_t *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.293 __bang_int42uchar_dn

```
void __bang_int42uchar_dn(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.294 __bang_int42uchar_oz

```
void __bang_int42uchar_oz(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.295 __bang_int42uchar_rd

```
void __bang_int42uchar_rd(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.296 __bang_int42uchar_rm

```
void __bang_int42uchar_rm(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.297 __bang_int42uchar_rn

```
void __bang_int42uchar_rn(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.298 __bang_int42uchar_tz

```
void __bang_int42uchar_tz(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.299 __bang_int42uchar_up

```
void __bang_int42uchar_up(unsigned char *dst,
                           int4x2_t *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `int4x2_t` to `unsigned char` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(unsigned char *dst, int4x2_t *src) {
    __nram__ int4x2_t src_nram[LEN];
    __nram__ unsigned char dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int4x2_t), GDRAM2NRAM);
    __bang_int42uchar_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(unsigned char), NRAM2GDRAM);
}
```

3.21.300 __bang_int82bfloat16

```
void __bang_int82bfloat16(bfloat16_t *dst,
                           int8_t *src,
                           int fix_position,
                           int count)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8 src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.301 __bang_int82bfloat16_dn

```
void __bang_int82bfloat16_dn(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 592;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq `mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.302 __bang_int82bfloat16_oz

```
void __bang_int82bfloat16_oz(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from int8_t to bfloat16_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.303 __bang_int82bfloat16_rd

```
void __bang_int82bfloat16_rd(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.304 __bang_int82bfloat16_rm

```
void __bang_int82bfloat16_rm(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 592;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq `mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.305 __bang_int82bfloat16_rn

```
void __bang_int82bfloat16_rn(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.306 __bang_int82bfloat16_tz

```
void __bang_int82bfloat16_tz(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.307 __bang_int82bfloat16_up

```
void __bang_int82bfloat16_up(bfloat16_t *dst,
                             int8_t *src,
                             int count,
                             int fix_position)
```

This function converts type of <src> from `int8_t` to `bfloat16_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82bfloat16_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.308 __bang_int82float

```
void __bang_int82float(float *dst,
                      int8_t *src,
                      int src_count,
                      int fix_position)
```

This function converts type of <src> from `int8_t` to `float` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- <fix_position> must be in the range [-127, 127];
- <src> cannot be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82float(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}
```

3.21.309 __bang_int82half

```
void __bang_int82half(half *dst,
                      int8_t *src,
                      int src_count,
                      int fix_position)

void __bang_int82half(half *dst,
                      int8_t *src,
                      int src_count,
                      int fix_position,
                      int dst_stride,
                      int src_stride,
                      int segnum)
```

This function converts type of `<src>` from `int8_t` to `half` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements according to `<fix_position>` in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{<fix_position>}$.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<dst_stride>` must be greater than or equal to `<src_count> * sizeof(half)`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(int8_t)`;
- $<src_stride> \div sizeof(int8_t) \geq <src_count>$ if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` cannot be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20
#define POS 5

__mlu_entry__ void kernel(half *dst, int8_t *src, int size) {
    __nram__ int8_t src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ half dst_tmp[DST_STRIDE / sizeof(half) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size, GDRAM2NRAM);
    __bang_int82half(dst_tmp, src_tmp, LEN, POS, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(half)), NRAM2GDRAM);
}
```

Cambricon@155chb

3.21.310 __bang_int82int16

```
void __bang_int82int16(int16_t *dst,
                      int8_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of `<src>` from `int8_t` to `int16_t` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_position>` and `<dst_position>` must be in the range [-127,127];

- <src> cannot be overlapped with <dst>;
- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int16_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int16(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

Cambricon@155chb

3.21.311 __bang_int82int32

```
void __bang_int82int32(int32_t *dst,
                      int8_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of <src> from int8_t to int32_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.
- [in] dst_position: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] src_position: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <src_position> and <dst_position> must be in the range [-127,127];
- <src> cannot be overlapped with <dst>;

- <src_count> must be greater than zero.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int32_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int32(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.312 __bang_int82int4_dn

```
void __bang_int82int4_dn(int4x2_t *dst,
                         int8_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from `int8_t` to `int4` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int4_dn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.313 __bang_int82int4_oz

Cambricon@155chb

```
void __bang_int82int4_oz(int4x2_t *dst,
                         int8_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of `<src>` from `int8_t` to `int4` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int4_oz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.314 __bang_int82int4_rd

```
void __bang_int82int4_rd(int4x2_t *dst,
                         int8_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

Cambricon@155chb

This function converts type of `<src>` from `int8_t` to `int4` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int4_rd(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.315 __bang_int82int4_rm

```
void __bang_int82int4_rm(int4x2_t *dst,
                         int8_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of `<src>` from `int8_t` to `int4` element-wisely in round-math mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<size>` must be greater than zero and divisible by 2;
- `<src>` can be overlapped with `<dst>`;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int4_rm(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.316 __bang_int82int4_rn

```
void __bang_int82int4_rn(int4x2_t *dst,
                         int8_t *src,
                         int size,
                         int dst_position,
                         int src_position)
```

This function converts type of <src> from `int8_t` to `int4` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127,127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>
```

```
#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int4_rn(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}
```

3.21.317 __bang_int82int4_tz

```
void __bang_int82int4_tz(int4x2_t *dst,
                          int8_t *src,
                          int size,
                          int dst_position,
                          int src_position)
```

This function converts type of <src> from `int8_t` to `int4` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127,127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.7.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
```

```

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int4x2_t dst_tmp[SIZE/2];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82int4_tz(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}

```

3.21.318 __bang_int82int4_up

```

void __bang_int82int4_up(int4x2_t *dst,
                         int8_t *src,
                         int size,
                         int dst_position,
                         int src_position)

```

This function converts type of <src> from `int8_t` to `int4` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `size`: The elements number of conversion.
- [in] `dst_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] `src_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <size> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <src_position> and <dst_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.7.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```

#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(int4x2_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];

```

```

__nram__ int4x2_t dst_tmp[SIZE/2];
__memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
__bang_int82int4_up(dst_tmp, src_tmp, SIZE, DSTPOS, SRCPOS);
__memcpy(dst, dst_tmp, SIZE * sizeof(int4x2_t) / 2, NRAM2GDRAM);
}

```

3.21.319 __bang_int82tf32

```

void __bang_int82tf32(float *dst,
                      int8_t *src,
                      int count,
                      int fix_position)

```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

Cambricon@155chb

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```

#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82tf32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}

```

3.21.320 __bang_int82tf32_dn

```
void __bang_int82tf32_dn(float *dst,
                          int8_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82tf32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.321 __bang_int82tf32_oz

```
void __bang_int82tf32_oz(float *dst,
                          int8_t *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.10`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int82tf32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.322 __bang_int8tf32_rd

```
void __bang_int8tf32_rd(float *dst,
                        int8_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int8tf32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.323 __bang_int8tf32_rm

```
void __bang_int8tf32_rm(float *dst,
                        int8_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int8tf32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.324 __bang_int8tf32_rn

```
void __bang_int8tf32_rn(float *dst,
                        int8_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int8tf32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.325 __bang_int8tf32_tz

```
void __bang_int8tf32_tz(float *dst,
                        int8_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `int8_t` to `tf32` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $\langle \text{src} \rangle \times 2^{\langle \text{fix_position} \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.10;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int8tf32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.326 __bang_int8tf32_up

```
void __bang_int8tf32_up(float *dst,
                        int8_t *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from int8_t to tf32 element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of source vector, i.e., $<src> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(float *dst, int8_t *src) {
    __nram__ int8_t src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(int8_t), GDRAM2NRAM);
    __bang_int8tf32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.327 __bang_short2half

```
void __bang_short2half(half *dst,
                      short *src,
                      int src_count)
```

```
void __bang_short2half(half *dst,
                      short *src,
                      int src_count,
                      int dst_stride,
                      int src_stride,
                      int segnum)
```

This function converts type of `<src>` from `short` to `half` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 64 on `(m)tp_2xx`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to `<src_count> * sizeof(half)`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- `<src_stride>` must be greater than or equal to zero, and divisible by `sizeof(short)`;
- $\frac{<src_stride>}{sizeof(short)} \geq <src_count>$ if `<src_stride>` is greater than zero;
- `<segnum>` must be greater than or equal to zero;
- `<src>` can be overlapped with `<dst>`.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has `<src_stride>` and `<dst_stride>`;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20

__mlu_entry__ void kernel(half *dst, short *src, int size) {
    __nram__ short src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ half dst_tmp[DST_STRIDE / sizeof(half) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size * sizeof(short), GDRAM2NRAM);
    __bang_short2half(dst_tmp, src_tmp, LEN, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(half)), NRAM2GDRAM);
}
```

3.21.328 __bang_tf322bfloat16

```
void __bang_tf322bfloat16(bfloat16_t *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to bfloat16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.329 __bang_tf322bfloat16_dn

```
void __bang_tf322bfloat16_dn(bfloat16_t *dst,
                             float *src,
                             int count)
```

This function converts type of <src> from tf32 to bfloat16_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
```

```
}
```

3.21.330 __bang_tf322bfloat16_oz

```
void __bang_tf322bfloat16_oz(bfloat16_t *dst,
                               float *src,
                               int count)
```

This function converts type of <src> from tf32 to bfloat16_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.331 __bang_tf322bfloat16_rd

```
void __bang_tf322bfloat16_rd(bfloat16_t *dst,
                             float *src,
                             int count)
```

This function converts type of <src> from tf32 to bfloat16_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16_rd(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_), NRAM2GDRAM);
}
```

3.21.332 __bang_tf322bfloat16_rm

```
void __bang_tf322bfloat16_rm(bfloat16_t *dst,
                             float *src,
                             int count)
```

This function converts type of <src> from tf32 to bfloat16_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.333 __bang_tf322bfloat16_rn

```
void __bang_tf322bfloat16_rn(bfloat16_t *dst,
                               float *src,
                               int count)
```

This function converts type of `<src>` from `tf32` to `bfloat16_t` element-wisely in round-nearest-even mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}
```

3.21.334 __bang_tf322bfloat16_tz

```
void __bang_tf322bfloat16_tz(bfloat16_t *dst,
                             float *src,
                             int count)
```

This function converts type of `<src>` from tf32 to bfloat16_t element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
```

```

__nram__ bfloat16_t dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
__bang_tf322bfloat16_tz(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}

```

3.21.335 __bang_tf322bfloat16_up

```

void __bang_tf322bfloat16_up(bfloat16_t *dst,
                             float *src,
                             int count)

```

This function converts type of <src> from tf32 to bfloat16_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(bfloat16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ bfloat16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322bfloat16_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}

```

3.21.336 __bang_tf322float

```
void __bang_tf322float(float *dst,
                      float *src,
                      int count)
```

This function converts type of <src> from tf32 to float element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.337 __bang_tf322float_dn

```
void __bang_tf322float_dn(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to float element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.338 __bang_tf322float_oz

```
void __bang_tf322float_oz(float *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `tf32` to `float` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.339 __bang_tf322float_rd

```
void __bang_tf322float_rd(float *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `tf32` to `float` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
```

```

__nram__ float dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
__bang_tf322float_rd(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}

```

3.21.340 __bang_tf322float_rm

```

void __bang_tf322float_rm(float *dst,
                           float *src,
                           int count)

```

This function converts type of <src> from tf32 to float element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}

```

3.21.341 __bang_tf322float_rn

```
void __bang_tf322float_rn(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to float element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.342 __bang_tf322float_tz

```
void __bang_tf322float_tz(float *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to float element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float_tz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.343 `__bang_tf322float_up`

```
void __bang_tf322float_up(float *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `tf32` to `float` element-wisely in round-up mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322float_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```

3.21.344 __bang_tf322half

```
void __bang_tf322half(half *dst,
                      float *src,
                      int count)
```

This function converts type of `<src>` from `tf32` to `half` element-wisely in round-to-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
```

```

__nram__ half dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
__bang_tf32half(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.345 __bang_tf32half_dn

```

void __bang_tf32half_dn(half *dst,
                        float *src,
                        int count)

```

This function converts type of <src> from tf32 to half element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32half_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.346 __bang_tf322half_oz

```
void __bang_tf322half_oz(half *dst,
                         float *src,
                         int count)
```

This function converts type of <src> from tf32 to half element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322half_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.347 __bang_tf322half_rd

```
void __bang_tf322half_rd(half *dst,
                         float *src,
                         int count)
```

This function converts type of <src> from tf32 to half element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32half_rd(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.348 __bang_tf32half_rm

```
void __bang_tf32half_rm(half *dst,
                        float *src,
                        int count)
```

This function converts type of `<src>` from `tf32` to `half` element-wisely in round-math mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32half_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.349 __bang_tf32half_rn

```
void __bang_tf32half_rn(half *dst,
                        float *src,
                        int count)
```

This function converts type of `<src>` from tf32 to half element-wisely in round-nearest-even mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 500;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_5xx.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
```

```

__nram__ half dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
__bang_tf32half_rn(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.350 __bang_tf32half_tz

```

void __bang_tf32half_tz(half *dst,
                        float *src,
                        int count)

```

This function converts type of <src> from tf32 to half element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32half_tz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}

```

3.21.351 __bang_tf32half_up

```
void __bang_tf32half_up(half *dst,
                        float *src,
                        int count)
```

This function converts type of <src> from tf32 to half element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(half *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ half dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32half_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(half), NRAM2GDRAM);
}
```

3.21.352 __bang_tf32int16

```
void __bang_tf32int16(int16_t *dst,
                      float *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from tf32 to int16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127,127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int16(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.353 __bang_tf32int16_dn

```
void __bang_tf32int16_dn(int16_t *dst,
                         float *src,
                         int count,
                         int fix_position)
```

This function converts type of <src> from tf32 to int16_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127,127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int16_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

Cambricon@155chb

3.21.354 __bang_tf32int16_oz

```
void __bang_tf32int16_oz(int16_t *dst,
                         float *src,
                         int count,
                         int fix_position)
```

This function converts type of <src> from tf32 to int16_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127,127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322int16_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.355 __bang_tf322int16_rd

```
void __bang_tf322int16_rd(int16_t *dst,
                           float *src,
                           int count,
                           int fix_position)
```

Cambricon@155chb

This function converts type of <src> from tf32 to int16_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., <dst> $\times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int16_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.356 __bang_tf32int16_rm

```
void __bang_tf32int16_rm(int16_t *dst,
                         float *src,
                         int count,
                         int fix_position)
```

This function converts type of <src> from tf32 to int16_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
```

```

__nram__ float src_nram[LEN];
__nram__ int16_t dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
__bang_tf32int16_rm(dst_nram, src_nram, LEN, FIX_POSITION);
__memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}

```

3.21.357 __bang_tf32int16_rn

```

void __bang_tf32int16_rn(int16_t *dst,
                          float *src,
                          int count,
                          int fix_position)

```

This function converts type of <src> from tf32 to int16_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

Cambricon@155chb

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127,127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```

#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int16_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}

```

3.21.358 __bang_tf32int16_tz

```
void __bang_tf32int16_tz(int16_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int16_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int16_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.359 __bang_tf32int16_up

```
void __bang_tf32int16_up(int16_t *dst,
                         float *src,
                         int count,
                         int fix_position)
```

This function converts type of <src> from tf32 to int16_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int16_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.360 __bang_tf32int32

```
void __bang_tf32int32(int32_t *dst,
                      float *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.361 __bang_tf32int32_dn

```
void __bang_tf32int32_dn(int32_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.362 __bang_tf32int32_oz

```
void __bang_tf32int32_oz(int32_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.363 __bang_tf32int32_rd

```
void __bang_tf32int32_rd(int32_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.364 __bang_tf32int32_rm

```
void __bang_tf32int32_rm(int32_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.365 __bang_tf32int32_rn

```
void __bang_tf32int32_rn(int32_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.366 __bang_tf32int32_tz

```
void __bang_tf32int32_tz(int32_t *dst,
                          float *src,
                          int count,
                          int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.367 __bang_tf32int32_up

```
void __bang_tf32int32_up(int32_t *dst,
                         float *src,
                         int count,
                         int fix_position)
```

This function converts type of <src> from tf32 to int32_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.368 __bang_tf32int4

```
void __bang_tf32int4(int4x2_t *dst,
                      float *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.369 __bang_tf32int4_dn

```
void __bang_tf32int4_dn(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.370 __bang_tf32int4_oz

```
void __bang_tf32int4_oz(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.371 __bang_tf32int4_rd

```
void __bang_tf32int4_rd(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., <dst> $\times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.372 __bang_tf32int4_rm

```
void __bang_tf32int4_rm(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.10;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.373 __bang_tf32int4_rn

```
void __bang_tf32int4_rn(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.374 __bang_tf32int4_tz

```
void __bang_tf32int4_tz(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_tz(dst_nram, src_nram, LEN ,FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.375 __bang_tf32int4_up

```
void __bang_tf32int4_up(int4x2_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int4x2_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int4_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int4x2_t), NRAM2GDRAM);
}
```

3.21.376 __bang_tf32int8

```
void __bang_tf32int8(int8_t *dst,
                      float *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.377 __bang_tf32int8_dn

```
void __bang_tf32int8_dn(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.378 __bang_tf32int8_oz

```
void __bang_tf32int8_oz(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.379 __bang_tf32int8_rd

```
void __bang_tf32int8_rd(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.380 __bang_tf32int8_rm

```
void __bang_tf32int8_rm(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.381 __bang_tf32int8_rn

```
void __bang_tf32int8_rn(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.382 __bang_tf32int8_tz

```
void __bang_tf32int8_tz(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.383 __bang_tf32int8_up

```
void __bang_tf32int8_up(int8_t *dst,
                        float *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from tf32 to int8_t element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.
- [in] fix_position: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int8_t *dst, float *src) {
    __nram__ float src_nram[LEN];
    __nram__ int8_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf32int8_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int8_t), NRAM2GDRAM);
}
```

3.21.384 __bang_tf322uchar

```
void __bang_tf322uchar(unsigned char *dst,
                      float *src,
                      int count)
```

This function converts type of <src> from tf32 to unsigned char element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 500;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_5xx.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.385 __bang_tf322uchar_dn

```
void __bang_tf322uchar_dn(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to unsigned char element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar_dn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.386 `__bang_tf322uchar_oz`

```
void __bang_tf322uchar_oz(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `tf32` to `unsigned char` element-wisely in round-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar_oz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.387 __bang_tf322uchar_rd

```
void __bang_tf322uchar_rd(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `tf32` to `unsigned char` element-wisely in round-nearest-off-zero mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
```

```

__nram__ float src_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
__bang_tf322uchar_rd(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}

```

3.21.388 __bang_tf322uchar_rm

```

void __bang_tf322uchar_rm(unsigned char *dst,
                           float *src,
                           int count)

```

This function converts type of <src> from tf32 to unsigned char element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar_rm(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}

```

3.21.389 __bang_tf322uchar_rn

```
void __bang_tf322uchar_rn(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to unsigned char element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] count: The number of elements.

Return

- void.

Remark

- <src> and <dst> must point to __nram__ address space;
- <count> must be greater than zero;
- <src> can be overlapped with <dst>.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 592;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_50;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_592.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar_rn(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.390 __bang_tf322uchar_tz

```
void __bang_tf322uchar_tz(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of <src> from tf32 to unsigned char element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592`.

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar_tz(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.391 __bang_tf322uchar_up

```
void __bang_tf322uchar_up(unsigned char *dst,
                           float *src,
                           int count)
```

This function converts type of `<src>` from `tf32` to `unsigned char` element-wisely in round-up mode and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` can be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592`;

- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(unsigned char *dst, float *src) {
    __nram__ unsigned char dst_nram[LEN];
    __nram__ float src_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(float), GDRAM2NRAM);
    __bang_tf322uchar_up(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.392 __bang_uchar2bfloat16

```
void __bang_uchar2bfloat16(bfloat16_t *dst,
                           unsigned char *src,
                           int count)
```

This function converts type of `<src>` from `unsigned char` to `bfloat16_t` element-wisely and saves the result in `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<count>` must be greater than zero;
- `<src>` cannot be overlapped with `<dst>`.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 592;`
- CNCC Version: `cncc --version >= 4.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_50;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_592.`

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(bfloat16_t *dst, unsigned char *src) {
```

```

__nram__ unsigned char src_nram[LEN];
__nram__ bfloat16_t dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
__bang_uchar2bfloat16(dst_nram, src_nram, LEN);
__memcpy(dst, dst_nram, LEN * sizeof(bfloat16_t), NRAM2GDRAM);
}

```

3.21.393 __bang_uchar2float

```

void __bang_uchar2float(float *dst,
                        unsigned char *src,
                        int count)

```

This function converts type of <src> from `unsigned char` to `float` element-wisely and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```

#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(unsigned char), GDRAM2NRAM);
    __bang_uchar2float(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}

```

3.21.394 __bang_uchar2half

```
void __bang_uchar2half(half *dst,
                      unsigned char *src,
                      int src_count)
```

```
void __bang_uchar2half(half *dst,
                      unsigned char *src,
                      int src_count,
                      int dst_stride,
                      int src_stride,
                      int segnum)
```

This function converts type of <src> from `unsigned char` to `half` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information. The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). <src> includes <segnum> + 1 blocks, and each block consists of <src_stride> bytes. <dst> includes <segnum> + 1 blocks, and each block consists of <dst_stride> bytes. In each block of <src>, this function converts first <src_count> elements in round-to-zero mode, and saves the result in blocks in <dst> sequentially. If <src_stride> is zero, this function only converts the first block <segnum> + 1 times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 128 on `(m)tp_2xx`, and divisible by 64 on `(m)tp_2xx`, and divisible by `sizeof(half)` on `(m)tp_3xx` or higher;
- <src_stride> must be greater than or equal to `<src_count> * sizeof(half)`, and divisible by `sizeof(unsigned char)`;
- <src> cannot be overlapped with <dst>;
- $\frac{<src_stride>}{\text{sizeof}(\text{unsignedchar})} \geq <\text{src_count}>$ if <src_stride> is greater than zero;
- <segnum> must be greater than or equal to zero.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;

- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20

__mlu_entry__ void kernel(half *dst, unsigned char *src, int size) {
    __nram__ unsigned char src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ half dst_tmp[DST_STRIDE / sizeof(half) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size, GDRAM2NRAM);
    __bang_uchar2half(dst_tmp, src_tmp, LEN, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(half)), NRAM2GDRAM);
}
```

3.21.395 __bang_uchar2int16

```
void __bang_uchar2int16(int16_t *dst,
                        unsigned char *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int16(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.396 __bang_uchar2int16_dn

```
void __bang_uchar2int16_dn(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

Cambricon@155chb

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 4.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_3xx.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int16_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.397 __bang_uchar2int16_oz

```
void __bang_uchar2int16_oz(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>
```

```
#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int16_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.398 __bang_uchar2int16_rd

```
void __bang_uchar2int16_rd(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
```

```

__nram__ unsigned char src_nram[LEN];
__nram__ int16_t dst_nram[LEN];
__memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
__bang_uchar2int16_rd(dst_nram, src_nram, LEN, FIX_POSITION);
__memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}

```

3.21.399 __bang_uchar2int16_rm

```

void __bang_uchar2int16_rm(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)

```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

Cambricon@155chb

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```

#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int16_rm(dst_nram, src_nram, LEN, FIX_POSITION);
}

```

```

    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}

```

3.21.400 __bang_uchar2int16_rn

```

void __bang_uchar2int16_rn(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)

```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```

#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int16_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}

```

3.21.401 __bang_uchar2int16_tz

```
void __bang_uchar2int16_tz(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(unsigned char), GDRAM2NRAM);
    __bang_uchar2int16_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.402 __bang_uchar2int16_up

```
void __bang_uchar2int16_up(int16_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int16_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int16_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int16_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int16_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int16_t), NRAM2GDRAM);
}
```

3.21.403 __bang_uchar2int32

```
void __bang_uchar2int32(int32_t *dst,
                        unsigned char *src,
                        int count,
                        int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.404 __bang_uchar2int32_dn

```
void __bang_uchar2int32_dn(int32_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.405 __bang_uchar2int32_oz

```
void __bang_uchar2int32_oz(int32_t *dst,
                            unsigned char *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.406 __bang_uchar2int32_rd

```
void __bang_uchar2int32_rd(int32_t *dst,
                            unsigned char *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.407 __bang_uchar2int32_rm

```
void __bang_uchar2int32_rm(int32_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.408 __bang_uchar2int32_rn

```
void __bang_uchar2int32_rn(int32_t *dst,
                            unsigned char *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.409 __bang_uchar2int32_tz

```
void __bang_uchar2int32_tz(int32_t *dst,
                            unsigned char *src,
                            int count,
                            int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.410 __bang_uchar2int32_up

```
void __bang_uchar2int32_up(int32_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int32_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int32_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int32_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int32_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(int32_t), NRAM2GDRAM);
}
```

3.21.411 __bang_uchar2int4

```
void __bang_uchar2int4(int4x2_t *dst,
                      unsigned char *src,
                      int count,
                      int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.412 __bang_uchar2int4_dn

```
void __bang_uchar2int4_dn(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-down mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_dn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.413 __bang_uchar2int4_oz

```
void __bang_uchar2int4_oz(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_oz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.414 __bang_uchar2int4_rd

```
void __bang_uchar2int4_rd(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-nearest-off-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_rd(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.415 __bang_uchar2int4_rm

```
void __bang_uchar2int4_rm(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-math mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_rm(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.416 __bang_uchar2int4_rn

```
void __bang_uchar2int4_rn(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-nearest-even mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_rn(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.417 __bang_uchar2int4_tz

```
void __bang_uchar2int4_tz(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(nt4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_tz(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.418 __bang_uchar2int4_up

```
void __bang_uchar2int4_up(int4x2_t *dst,
                           unsigned char *src,
                           int count,
                           int fix_position)
```

This function converts type of <src> from `unsigned char` to `int4x2_t` element-wisely in round-up mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <count> must be greater than zero and divisible by 2;
- <src> can be overlapped with <dst>;
- <fix_position> must be in the range [-127, 127].

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 4.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_3xx`.

Example

```
#include <bang.h>

#define LEN 128
#define FIX_POSITION 2

__mlu_entry__ void kernel(int4x2_t *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ int4x2_t dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2int4_up(dst_nram, src_nram, LEN, FIX_POSITION);
    __memcpy(dst, dst_nram, LEN * sizeof(char), NRAM2GDRAM);
}
```

3.21.419 __bang_uchar2tf32

```
void __bang_uchar2tf32(float *dst,
                        unsigned char *src,
                        int count)
```

This function converts type of <src> from `unsigned char` to `tf32` element-wisely and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <count> must be greater than zero;
- <src> cannot be overlapped with <dst>.

Requirements

- Cambricon BANG Version: `_BANG_ARCH_` \geq 500;
- CNCC Version: `cncc --version` \geq 4.0.0;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch` \geq `compute_50`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch` \geq (`m`)`tp_5xx`.

Example

Cambricon@155chb

```
#include <bang.h>

#define LEN 128

__mlu_entry__ void kernel(float *dst, unsigned char *src) {
    __nram__ unsigned char src_nram[LEN];
    __nram__ float dst_nram[LEN];
    __memcpy(src_nram, src, LEN * sizeof(char), GDRAM2NRAM);
    __bang_uchar2tf32(dst_nram, src_nram, LEN);
    __memcpy(dst, dst_nram, LEN * sizeof(float), NRAM2GDRAM);
}
```



4 Deprecated Built-in Functions

4.1 __bang_add_const

```
void __bang_add_const(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_add_const(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function adds `<value>` to `<elem_count>` elements of `<src>` and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `_nram_` address space;
- `<src>` can be overlapped with `<dst>`;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- This function was deprecated from CNCC v4.0.0. Use [`__bang_add_scalar`](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_add_const(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

4.2 __bang_band

```
void __bang_band(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)
```

```
void __bang_band(short *dst,
                  short *src0,
                  short *src1,
                  int elem_count)
```

Applies bit-wise AND operation on two vectors.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src0>;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- int and short are supported on (m)tp_3xx or higher;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use __bang_band instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;

- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(int* c, int* a, int* b) {
    __nram__ int a_tmp[DATA_SIZE];
    __nram__ int c_tmp[DATA_SIZE];
    __nram__ int b_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(int), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(int), GDRAM2NRAM);
    __bang_band(c_tmp, a_tmp, b_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(int), NRAM2GDRAM);
}
```

4.3 __bang_bnot

```
void __bang_bnot(int *dst,
                  int *src,
                  int elem_count)
void __bang_bnot(short *dst,
                  short *src,
                  int elem_count)
```

Applies bit-wise NOT operation on a vector.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- int and short are supported on (m)tp_3xx or higher;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use __bang_bnot instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;

- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

4.4 __bang_bor

```
void __bang_bor(int *dst,
                 int *src0,
                 int *src1,
                 int elem_count)
```

```
void __bang_bor(short *dst,
                 short *src0,
                 short *src1,
                 int elem_count)
```

Applies bit-wise OR operation on two vectors.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src0`: The address of first source vector.
- [in] `src1`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` can be overlapped with `<src0>`;
- `<src0>, <src1>` and `<dst>` must point to `__nram__` address space;
- `int` and `short` are supported on `(m)tp_3xx` or higher;
- `<elem_count> * sizeof(type)` must be divisible by 128 on `(m)tp_2xx`;
- The address of `<src0>, <src1>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- This function was deprecated from CNCC v4.0.0. Use `__bang_bor` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200;`
- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

```
void __bang_bxor(int *dst,
                  int *src0,
                  int *src1,
                  int elem_count)

void __bang_bxor(short *dst,
                  short *src0,
                  short *src1,
                  int elem_count)
```

Applies bit-wise XOR operation on two vectors.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of first source vector.
- [in] src1: The address of second source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> can be overlapped with <src0>;
- <src0>, <src1> and <dst> must point to __nram__ address space;
- int and short are supported on (m)tp_3xx or higher;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src0>, <src1> and <dst> must be 64-byte aligned on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_bxor](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.6 __bang_char2int

```
void __bang_char2int(int32_t *dst,
                      int8_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of <src> from `int8_t` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127,127];
- <src> cannot be overlapped with <dst>;
- <src_count> must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int82int32` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int32_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_char2int(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

4.7 __bang_char2short

```
void __bang_char2short(int16_t *dst,
                      int8_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of <src> from `int8_t` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127,127];
- <src> cannot be overlapped with <dst>;
- <src_count> must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int82int16` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int16_t *dst, int8_t *src) {
    __nram__ int8_t src_tmp[SIZE];
    __nram__ int16_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int8_t), GDRAM2NRAM);
    __bang_char2short(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}
```

4.8 __bang_count

```
void __bang_count(unsigned int *dst,
                  half *src,
                  int elem_count)
```

```
void __bang_count(unsigned int *dst,
                  float *src,
                  int elem_count)
```

Counts the number of non-zero elements in the input vector.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The size of <dst> is at least 128 bytes on (m)tp_2xx, and at least 4 bytes on (m)tp_3xx;
- The number of non-zero elements is stored in the first 4 bytes of <dst>. The remaining 124 bytes of <dst> on (m)tp_2xx will be set to 0. The remaining part of <dst> will not be changed on (m)tp_3xx;
- <dst> can be overlapped with <src>;
- This function was deprecated from CNCC v4.0.0. Use __bang_count instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(unsigned int* dst, float* src, int elem_count) {
    __bang_count(dst, src, elem_count);
    unsigned int counter = ((unsigned int*)dst)[0];
}
```

4.9 __bang_count_bitindex

```
void __bang_count_bitindex(unsigned int *dst,
                           half *src,
                           int elem_count)
```

```
void __bang_count_bitindex(unsigned int *dst,
                           float *src,
                           int elem_count)
```

Counts the number of non-zero bit in the input vector.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- The size of <dst> is at least 128 bytes on (m)tp_2xx, and at least 4 bytes on (m)tp_3xx;
- The number of non-zero bits is stored in the first 4 bytes of <dst>. The remaining 124 bytes of <dst> on (m)tp_2xx will be set to 0. The remaining part of <dst> will not be changed on (m)tp_3xx;
- <dst> can be overlapped with <src>;
- This function was deprecated from CNCC v4.0.0. Use __bang_count_bitindex instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(unsigned int* dst, float* src, int elem_count) {
    __bang_count_bitindex(dst, src, elem_count);
    unsigned int counter = ((unsigned int*)dst)[0];
}
```

```
void __bang_cycle_band(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_band(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts. Each element in each part applies bit-wise AND operation with the corresponding element in `<seg>`. The result is assigned to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `int` and `short` are supported on `(m)tp_3xx` or higher;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_elem_count> * sizeof(char)` and `<seg_elem_count> * sizeof(char)` must be divisible by 128 on `(m)tp_2xx`;
- This function was deprecated from CNCC v4.0.0. Use corresponding char function instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

4.11 __bang_cycle_bor

```
void __bang_cycle_bor(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

```
void __bang_cycle_bor(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides `<src>` into `<src_elem_count> / <seg_elem_count>` parts. Each element in each part applies bit-wise OR operation with the corresponding element in `<seg>`. The result is assigned to `<dst>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `seg`: The address of second source vector.
- [in] `src_elem_count`: The number of elements in `<src>` vector.
- [in] `seg_elem_count`: The number of elements in `<seg>` vector.

Return

- `void`.

Remark

- `<dst>` can be overlapped with `<src>`;
- `<src>`, `<seg>` and `<dst>` must point to `__nram__` address space;
- `int` and `short` are supported on `(m)tp_3xx` or higher;
- `<src_elem_count>` must be divisible by `<seg_elem_count>`;
- `<src_elem_count>` and `<seg_elem_count>` must be greater than zero;
- The address of `<src>`, `<seg>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_elem_count> * sizeof(char)` and `<seg_elem_count> * sizeof(char)` must be divisible by 128 on `(m)tp_2xx`;
- This function was deprecated from CNCC v4.0.0. Use corresponding char function instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

```
void __bang_cycle_bxor(int *dst,
                      int *src,
                      int *seg,
                      int src_elem_count,
                      int seg_elem_count)

void __bang_cycle_bxor(short *dst,
                      short *src,
                      short *seg,
                      int src_elem_count,
                      int seg_elem_count)
```

Divides <src> into <src_elem_count> / <seg_elem_count> parts. Each element in each part applies bit-wise XOR operation with the corresponding element in <seg>. The result is assigned to <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] seg: The address of second source vector.
- [in] src_elem_count: The number of elements in <src> vector.
- [in] seg_elem_count: The number of elements in <seg> vector.

Return

- void.

Remark

- <dst> can be overlapped with <src>;
- <src>, <seg> and <dst> must point to __nram__ address space;
- int and short are supported on (m)tp_3xx or higher;
- <src_elem_count> must be divisible by <seg_elem_count>;
- <src_elem_count> and <seg_elem_count> must be greater than zero;
- The address of <src>, <seg> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <src_elem_count> * sizeof(char) and <seg_elem_count> * sizeof(char) must be divisible by 128 on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_cycle_bxor](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.13 __bang_div

```
void __bang_div(half *dst,
                 half *src0,
                 half *src1,
                 half *src_addition,
                 int elem_count)
```

```
void __bang_div(float *dst,
                 float *src0,
                 float *src1,
                 float *src_addition,
                 int elem_count)
```

This function performs division operation element-wisely on <src0> and <src1> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Div Operation Function](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src0: The address of the first source vector.
- [in] src1: The address of the second source vector.
- [in] src_addition: The address of additional vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <src0>, <src1>, <dst> and <src_addition> must point to __nram__ address space;
- <src0>, <src1>, <dst> and <src_addition> cannot be overlapped;
- <elem_count> * sizeof(type) must be divisible by 128;
- The size of <src_addition> vector is identical to the size of <src0>, <src1> and <dst>;
- <elem_count> must be greater than zero;
- For higher precision, use surpass function [__bang_div](#) on tp_322 instead;
- For higher precision, use [__bang_recip](#) and [__bang_mul](#) on mtp_372 instead;
- This function was deprecated from CNCC v4.0.0. Use [__bang_div](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128
```

```

__mlu_entry__ void kernel(float* c, float* a, float* b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __nram__ float b_tmp[DATA_SIZE];
    __nram__ float addition[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __memcpy(b_tmp, b, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_div(c_tmp, a_tmp, b_tmp, src_addition, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}

```

4.14 __bang_findfirst1

```
void __bang_findfirst1(unsigned int *dst,
                      half *src,
                      int elem_count)
```

```
void __bang_findfirst1(unsigned int *dst,
                      float *src,
                      int elem_count)
```

Finds the first non-zero data in the values of <src>, and stores the index of the first non-zero data in the first element of <dst>. If <src> is all zero, stores 0xffff-ffff-ffff-ffff.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <elem_count> * sizeof (type) must be divisible by 128 on (m)tp_2xx;
- The total number of bytes of <dst> is at least 128 on (m)tp_2xx, and at least 4 on (m)tp_3xx;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>;
- This function was deprecated from CNCC v4.0.0. Use [__bang_findfirst1](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;

- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 64

__mlu_entry__ void kernel(half* c, half* a) {
    __nram__ half a_tmp[DATA_SIZE];
    __nram__ half c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(half), GDRAM2NRAM);
    __bang_findfirst1((uint32_t*)c_tmp, a_tmp, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(half), NRAM2GDRAM);
}
```

4.15 __bang_findlast1

```
void __bang_findlast1(unsigned int *dst,
                      half *src,
                      int elem_count)
```

```
void __bang_findlast1(unsigned int *dst,
                      float *src,
                      int elem_count)
```

Finds the last non-zero data in the values of <src>, and stores the index of the last non-zero data in <dst>. If <src> is all zero, stores 0xffff-ffff-ffff-ffff.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <elem_count> * sizeof (type) must be divisible by 128 on (m)tp_2xx;
- The total number of bytes of <dst> is at least 128 on (m)tp_2xx, and at least 4 on (m)tp_3xx;
- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <dst> can be overlapped with <src>;
- This function was deprecated from CNCC v4.0.0. Use __bang_findlast1 instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;

- CNCC Version: `cncc --version >= 2.8.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

4.16 __bang_fix82half

```
void __bang_fix82half(half *dst,
                      int8_t *src,
                      int src_count,
                      int fix_position)
```

```
void __bang_fix82half(half *dst,
                      int8_t *src,
                      int src_count,
                      int fix_position,
                      int dst_stride,
                      int src_stride,
                      int segnum)
```

This function converts type of `<src>` from `int8_t` to `half` element-wisely in round-to-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

The data type conversion process is illustrated by Figure [The Process of Conversion With Stride](#). `<src>` includes `<segnum> + 1` blocks, and each block consists of `<src_stride>` bytes. `<dst>` includes `<segnum> + 1` blocks, and each block consists of `<dst_stride>` bytes. In each block of `<src>`, this function converts first `<src_count>` elements according to `<fix_position>` in round-to-zero mode, and saves the result in blocks in `<dst>` sequentially. If `<src_stride>` is zero, this function only converts the first block `<segnum> + 1` times.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `fix_position`: Scale factor of source vector, i.e., $<src> \times 2^{<fix_position>}.$
- [in] `dst_stride`: The destination stride in bytes.
- [in] `src_stride`: The source stride in bytes.
- [in] `segnum`: The number of segments minus one.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<segnum>` must be greater than or equal to zero;
- `<src>` cannot be overlapped with `<dst>`;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;

- <src_count> must be greater than zero and divisible by 128 on (m)tp_2xx;
- <fix_position> must be in the range [-127, 127];
- <dst_stride> must be greater than or equal to <src_count> * sizeof(half), and divisible by 64 on (m)tp_2xx, and divisible by sizeof(half) on (m)tp_3xx or higher;
- <src_stride> must be greater than or equal to zero, and divisible by sizeof(int8_t);
- <src_stride> \div sizeof(int8_t) \geq <src_count> if <src_stride> is greater than zero;
- This function was deprecated from CNCC v4.0.0. Use [__bang_int82half](#) instead.

Instruction Pipeline

- Execute in Move instruction pipeline if the conversion function has <src_stride> and <dst_stride>;
- Execute in Compute instruction pipeline, otherwise.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ \geq 200;
- CNCC Version: cncc --version \geq 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch \geq compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch \geq (m)tp_2xx.

Example

```
#include <bang.h>

#define SRC_STRIDE 160
#define DST_STRIDE 320
#define LEN 128
#define SEG_NUM 20
#define POS 5

__mlu_entry__ void kernel(half *dst, int8_t *src, int size) {
    __nram__ int8_t src_tmp[SRC_STRIDE * SEG_NUM + LEN];
    __nram__ half dst_tmp[DST_STRIDE / sizeof(half) * SEG_NUM + LEN];
    __memcpy(src_tmp, src, size, GDRAM2NRAM);
    __bang_fix82half(dst_tmp, src_tmp, LEN, POS, DST_STRIDE, SRC_STRIDE, SEG_NUM);
    __memcpy(dst, dst_tmp, (DST_STRIDE * SEG_NUM + LEN * sizeof(half)), NRAM2GDRAM);
}
```

Cambricon@155chb

4.17 __bang_float2int_dn

```
void __bang_float2int_dn(int32_t *dst,
                        float *src,
                        int src_count,
                        int fix_position)
```

This function converts type of <src> from float to int32_t element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.

- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{fix_position}$.

Return

- void.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`;
- This function was deprecated from CNCC v4.0.0. Use `__bang_float2int32_dn` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(int32_t *dst, float *src) {
    __nram__ float src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(float), GDRAM2NRAM);
    __bang_float2int_dn(dst_tmp, src_tmp, SIZE, POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

4.18 __bang_float2int_oz

```
void __bang_float2int_oz(int32_t *dst,
                        float *src,
                        int src_count,
                        int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-off-zero mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.

- [in] fix_position: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{fix_position}$.

Return

- void.

Remark

- $\langle src \rangle$ and $\langle dst \rangle$ must point to `__nram__` address space;
- $\langle src_count \rangle$ must be greater than zero;
- $\langle fix_position \rangle$ must be in the range [-127, 127];
- $\langle src \rangle$ can be overlapped with $\langle dst \rangle$;
- This function was deprecated from CNCC v4.0.0. Use `__bang_float2int32_oz` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of `__bang_float2int_dn` for more details.

4.19 __bang_float2int_rd

```
void __bang_float2int_rd(int32_t *dst,
                         float *src,
                         int src_count,
                         int fix_position)
```

This function converts type of $\langle src \rangle$ from `float` to `int32_t` element-wisely in round-nearest-off-zero mode and saves the result in $\langle dst \rangle$. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The elements number of conversion.
- [in] fix_position: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{fix_position}$.

Return

- void.

Remark

- $\langle src \rangle$ and $\langle dst \rangle$ must point to `__nram__` address space;
- $\langle src_count \rangle$ must be greater than zero;
- $\langle fix_position \rangle$ must be in the range [-127, 127];
- $\langle src \rangle$ can be overlapped with $\langle dst \rangle$;
- This function was deprecated from CNCC v4.0.0. Use `__bang_float2int32_rd` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;

- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of [__bang_float2int_dn](#) for more details.

4.20 __bang_float2int_rm

```
void __bang_float2int_rm(int32_t *dst,
                         float *src,
                         int src_count,
                         int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-math mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Cambricon@155chb

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range $[-127, 127]$;
- `<src>` can be overlapped with `<dst>`;
- This function was deprecated from CNCC v4.0.0. Use [__bang_float2int32_rm](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

- See the example of [__bang_float2int_dn](#) for more details.

4.21 __bang_float2int_rn

```
void __bang_float2int_rn(int32_t *dst,
                         float *src,
                         int src_count,
                         int fix_position)
```

This function converts type of <src> from `float` to `int32_t` element-wisely in round-nearest-even mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{fix_position}$.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- This function was deprecated from CNCC v4.0.0. Use [__bang_float2int32_rn](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of [__bang_float2int_dn](#) for more details.

4.22 __bang_float2int_tz

```
void __bang_float2int_tz(int32_t *dst,
                         float *src,
                         int src_count,
                         int fix_position)
```

This function converts type of <src> from `float` to `int32_t` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.

- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`;
- This function was deprecated from CNCC v4.0.0. Use `__bang_float2int32_tz` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

- See the example of `__bang_float2int_dn` for more details.

4.23 __bang_float2int_up

```
Cambricon@155chb
void __bang_float2int_up(int32_t *dst,
                         float *src,
                         int src_count,
                         int fix_position)
```

This function converts type of `<src>` from `float` to `int32_t` element-wisely in round-up mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle fix_position \rangle}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_count>` must be greater than zero;
- `<fix_position>` must be in the range [-127, 127];
- `<src>` can be overlapped with `<dst>`;
- This function was deprecated from CNCC v4.0.0. Use `__bang_float2int32_up` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 322;
- CNCC Version: cncc --version >= 3.0.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_30;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= tp_322.

Example

- See the example of [__bang_float2int_dn](#) for more details.

4.24 __bang_ge_const

```
void __bang_ge_const(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_ge_const(float *dst,
                     float *src,
                     float value,
                     int elem_count)
```

This function compares `<elem_count>` elements in `<src>` with `<value>` to determine whether the elements are greater than or equal to `<value>` and saves the result in `<dst>`. If the element of `<src>` is greater than or equal to `<value>`, the result is 1.0. Otherwise, the result is 0.0. The type of result is same as the type of the element of `<src>`. See the table [Floating Point Calculation of Stream and Scalar Comparison Functions](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `value`: The source scalar.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src>` can be overlapped with `<dst>`;
- `<elem_count> * sizeof(type)` must be divisible by 128 on (m)tp_2xx;
- The address of `<src>` and `<dst>` must be 64-byte aligned on (m)tp_2xx;
- This function is deprecated since 4.0.0, please use [__bang_ge_scalar](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include <bang.h>

#define DATA_SIZE 128

__mlu_entry__ void kernel(float* c, float* a, float b) {
    __nram__ float a_tmp[DATA_SIZE];
    __nram__ float c_tmp[DATA_SIZE];
    __memcpy(a_tmp, a, DATA_SIZE * sizeof(float), GDRAM2NRAM);
    __bang_ge_const(c_tmp, a_tmp, b, DATA_SIZE);
    __memcpy(c, c_tmp, DATA_SIZE * sizeof(float), NRAM2GDRAM);
}
```

4.25 __bang_half2char_dn

```
void __bang_half2char_dn(signed char *dst,
                         half *src,
                         int src_count)
```

This function converts type of `<src>` from `half` to `signed char` element-wisely in round-down mode and saves the result in `<dst>`. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<src_count>` must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- `<src>` can be overlapped with `<dst>`;
- This function was deprecated from CNCC v4.0.0. Use [__bang_half2uchar_dn](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128
```

```
__mlu_entry__ void kernel(signed char *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ signed char dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
    __bang_half2char_dn(dst_tmp, src_tmp, SIZE);
    __memcpy(dst, dst_tmp, SIZE * sizeof(signed char), NRAM2GDRAM);
}
```

4.26 __bang_half2uchar_dn

```
void __bang_half2uchar_dn(signed char *dst,
                           half *src,
                           int src_count)
```

This function converts type of <src> from `half` to `unsigned char` element-wisely in round-down mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Integer](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.

Return

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- The address of <src> and <dst> must be 64-byte aligned on `(m)tp_2xx`;
- <src_count> must be greater than zero and divisible by 128 on `(m)tp_2xx`;
- <src> can be overlapped with <dst>;
- This function was deprecated from CNCC v4.0.0. Use [__bang_half2uchar_dn](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

```
#include <bang.h>

#define SIZE 128

__mlu_entry__ void kernel(signed char *dst, half *src) {
    __nram__ half src_tmp[SIZE];
    __nram__ signed char dst_tmp[SIZE];
```

```

__memcpy(src_tmp, src, SIZE * sizeof(half), GDRAM2NRAM);
__bang_half2uchar_dn(dst_tmp, src_tmp, SIZE);
__memcpy(dst, dst_tmp, SIZE * sizeof(unsigned char), NRAM2GDRAM);
}

```

4.27 __bang_int2char

```

void __bang_int2char(int8_t *dst,
                     int32_t *src,
                     int src_count,
                     int dst_position,
                     int src_position)

```

This function converts type of <src> from `int32_t` to `int8_t` element-wisely round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

Cambricon@155chb

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_position> and <dst_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int32int8` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```

#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int8_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];

```

```

__nram__ int8_t dst_tmp[SIZE];
__memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
__bang_int2char(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
__memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}

```

4.28 __bang_int2float

```

void __bang_int2float(float *dst,
                      int32_t *src,
                      int src_count,
                      int fix_position)

```

This function converts type of <src> from `int32_t` to `float` element-wisely in round-to-zero mode and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point](#) for accuracy information.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The elements number of conversion.
- [in] `fix_position`: Scale factor of source vector, i.e., $<\text{src}> \times 2^{<\text{fix_position}>}$.

Return

Cambricon@155chb

- `void`.

Remark

- <src> and <dst> must point to `__nram__` address space;
- <src_count> must be greater than zero;
- <fix_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int322float` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```

#include <bang.h>

#define SIZE 128
#define POS 5

__mlu_entry__ void kernel(float *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];
    __nram__ float dst_tmp[SIZE];

```

```

__memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
__bang_int2float(dst_tmp, src_tmp, SIZE, POS);
__memcpy(dst, dst_tmp, SIZE * sizeof(float), NRAM2GDRAM);
}

```

4.29 __bang_int2short

```

void __bang_int2short(int16_t *dst,
                      int32_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)

```

This function converts type of <src> from `int32_t` to `int16_t` element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}.$
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}.$

Return

Cambricon@155chb

- `void`.

Remark

- <src> and <dst> must point to `_nram_` address space;
- <src_position> and <dst_position> must be in the range [-127, 127];
- <src> can be overlapped with <dst>;
- <src_count> must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int32int16` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322;`
- CNCC Version: `cncc --version >= 3.0.0;`
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30;`
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322.`

Example

```

#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int16_t *dst, int32_t *src) {
    __nram__ int32_t src_tmp[SIZE];

```

```

__nram__ int16_t dst_tmp[SIZE];
__memcpy(src_tmp, src, SIZE * sizeof(int32_t), GDRAM2NRAM);
__bang_int2short(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
__memcpy(dst, dst_tmp, SIZE * sizeof(int16_t), NRAM2GDRAM);
}

```

4.30 __bang_lock

```
void __bang_lock(int lock_id_0,
                  int lock_id_1)
```

Applies and seize lock_id.

Parameters

- [in] lock_id_0: The first lock id.
- [in] lock_id_1: The second lock id.

Return

- void.

Remark

- This function was deprecated from CNCC v4.0.0;
- [__bang_lock](#) is always paired with [__bang_unlock](#);
- For MLU Core, lock_id_0 should be equal to lock_id_1;
- For MPU Core, lock_id_0 should be not equal to lock_id_1;
- [__bang_lock](#) can be used before [__memcpy](#) from to GDRAM, whose size is larger than 64KB.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= mtp_2xx`.

Example

Example of [__bang_lock](#) usage in MLU Core is as follows:

```

#include <bang.h>

#define BUFFER_SIZE 261632

__mlu_entry__ void add_kernel(half *input, half *output,
                             int copyin_len, int copyout_len,
                             int offseta, int offsetb,
                             int offsetc, int tid) {
    __wram__ half temp[BUFFER_SIZE];
    if (coreId != 0x80) {
        __bang_lock(0, 0);
    }
}
```

```

    }
    __memcpy(temp, input, 510 * 512 * sizeof(half), GDRAM2WRAM);
    if (coreId != 0x80) {
        __bang_unlock(0, 0);
    }
}

```

4.31 __bang_maskmove

```

void __bang_maskmove(half *dst,
                     half *src,
                     half *mask,
                     int elem_count)

void __bang_maskmove(float *dst,
                     float *src,
                     float *mask,
                     int elem_count)

```

Selects bytes in `<src>`, whose element count is `<elem_count>`, according to the bit value of the vector `<mask>`, and stores the result in `<dst>`. The bytes in `<src>` will be selected if corresponding bit values in `<mask>` are not equal to zero. If the bit value of mask is 1, stores the corresponding byte of `<src>` to `<dst>`; otherwise, keeps the corresponding byte in `<dst>` unchanged.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of source vector.
- [in] `mask`: The address of mask vector.
- [in] `elem_count`: The length of source vector.

Return

- `void`.

Remark

- `<elem_count> * sizeof(type)` must be a multiple of 1024 bytes on `(m)tp_2xx`;
- `<elem_count> * sizeof(type)` must be a multiple of 8 bytes on `(m)tp_3xx`;
- `<src>`, `<mask>` and `<dst>` must point to `__nram__` address space;
- The address of `<src>`, `<mask>` and `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<dst>` can be overlapped with `<src>`;
- `<elem_count>` must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_maskmove` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;

- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.32 __bang_maskmove_bitindex

```
void __bang_maskmove_bitindex(half *dst,
                             half *src,
                             void *bitmask,
                             int elem_count)

void __bang_maskmove_bitindex(float *dst,
                             float *src,
                             void *bitmask,
                             int elem_count)
```

Selects the corresponding elements in `<src>` according to `<bitmask>`. The elements in `<src>` will be saved to `<dst>`, if corresponding bit in `<bitmask>` is 1. All selected elements in `<dst>` will be stored continuously.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `src`: The address of first source vector.
- [in] `bitmask`: The address of second source vector.
- [in] `elem_count`: The number of elements in source vector.

Return

- `void`.

Remark

- `<elem_count>` must be greater than zero;
- `<dst>` can be overlapped with `<src>`;
- The `<src>`, `<bitmask>` and `<dst>` must point to `__nram__` address space;
- The total number of bytes of `<dst>` is at least 128 on (m)tp_2xx;
- The address of `<src>`, `<bitmask>` and `<dst>` must be 64-byte aligned on (m)tp_2xx;
- `<elem_count>` must be divisible by 1024 on (m)tp_2xx, and divisible by 8 on mtp_372 and tp_322;
- This function was deprecated from CNCC v4.0.0. Use `__bang_collect_bitindex` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.8.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

4.33 __bang_maximum

```
void __bang_maximum(half *dst,
                    half *src,
                    int distance,
                    int size)
```

Finds maximum value of each two corresponding elements in two vectors between which the distance is <distance>. See the table [Element-wise Floating Point Calculation of Stream and Scalar Comparison Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of first source vector.
- [in] distance: The distance between the two source vector.
- [in] size: The elements number of destination vector.

Return

- void.

Remark

- <size> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> cannot be overlapped with <src>;
- <size> * sizeof (type) must be divisible by 128 on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use __bang_maxequal instead.

Instruction Pipeline

- Compute.

Requirements

- None.

Example

4.34 __bang_mul_const

```
void __bang_mul_const(half *dst,
                      half *src,
                      half value,
                      int elem_count)
```

```
void __bang_mul_const(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function multiplies <elem_count> elements of <src> by <value> and saves the result in <dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <dst> can be overlapped with <src>;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_mul_scalar](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_add_const](#) for more details.

Cambricon@155chb

4.35 __bang_pad

```
void __bang_pad(half *dst,
                 half *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(short *dst,
                 short *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(unsigned short *dst,
                unsigned short *src,
                int channel,
                int height,
                int width,
                int pad_height,
                int pad_width)
```

```
void __bang_pad(int8_t *dst,
                 int8_t *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(char *dst,
                 char *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(unsigned char *dst,
                 unsigned char *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(float *dst,
                 float *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(int *dst,
                int *src,
                int channel,
                int height,
                int width,
                int pad_height,
                int pad_width)
```

```
void __bang_pad(unsigned int *dst,
                 unsigned int *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width)
```

```
void __bang_pad(half *dst,
                 half *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width,
                 half pad_value)
```

```
void __bang_pad(short *dst,
                 short *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width,
                 short pad_value)
```

```
void __bang_pad(char *dst,
                 char *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width,
                 char pad_value)
```

```
void __bang_pad(float *dst,
                 float *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width,
                 float pad_value)
```

```
void __bang_pad(int *dst,
                 int *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width,
                 int pad_value)
```

```
void __bang_pad(bfloat16_t *dst,
                 bfloat16_t *src,
                 int channel,
                 int height,
                 int width,
                 int pad_height,
                 int pad_width,
                 bfloat16_t pad_value)
```

Applies padding operation on <src>.

Parameters

- [out] dst: The destination vector, whose data layout is HWC.
- [in] src: The source vector, whose data layout is HWC.
- [in] channel: Number of channels.
- [in] height: The height of <src>.
- [in] width: The width of <src>.
- [in] pad_width: Number of columns whose elements is all zero or <pad_value> on the horizontal of pad.
- [in] pad_height: Number of columns whose elements is all zero or <pad_value> on the vertical of pad.
- [in] pad_value: The value of padding.

Return

- void.

Remark

- <height> and <width> must be greater than 0;
- bfloat16_t is supported on (m)tp_5xx or higher;
- <dst> cannot be overlapped with <src>;

- <src> and <dst> must point to __nram__ address space;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- <channel> * <width> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- <pad_width> * <channel> * sizeof(type) must be 128-byte aligned on (m)tp_2xx;
- If <height> == 1, <pad_width> must be equal to 0;
- <pad_value> are only supported on (m)tp_5xx or higher;
- This function was deprecated from CNCC v4.0.0. Use __bang_pad instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

```
#include<bang.h>

#define PAD_H 1
#define PAD_W 2
#define INPUT_H 4
#define INPUT_W 4
#define CHANNEL 64

__mlu_entry__ void kernel(half *out, half *in) {
    __nram__ half nx[CHANNEL * INPUT_H * INPUT_W];
    __nram__ half ny[CHANNEL * (INPUT_H + 2 * PAD_H) * (INPUT_W + 2 * PAD_W)];
    __memcpy(nx, in, CHANNEL * INPUT_H * INPUT_W * sizeof(half), GDRAM2NRAM);
    __bang_pad(ny, nx, CHANNEL, INPUT_H, INPUT_W, PAD_H, PAD_W);
    __memcpy(out, ny, CHANNEL * (INPUT_H + 2 * PAD_H) * (INPUT_W + 2 * PAD_W) * sizeof(half), NRAM2GDRAM);
}
```

4.36 __bang_short2char

```
void __bang_short2char(int8_t *dst,
                      int16_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of <src> from int16_t to int8_t element-wisely in round-to-zero mode and saves the result in <dst>.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] src_count: The number of elements.

- [in] dst_position: Scale factor of destination vector, i.e., $\langle dst \rangle \times 2^{\langle dst_position \rangle}$.
- [in] src_position: Scale factor of source vector, i.e., $\langle src \rangle \times 2^{\langle src_position \rangle}$.

Return

- void.

Remark

- $\langle src \rangle$ and $\langle dst \rangle$ must point to $__nram__$ address space;
- $\langle src_position \rangle$ and $\langle dst_position \rangle$ must be in the range [-127, 127];
- $\langle src \rangle$ can be overlapped with $\langle dst \rangle$;
- $\langle src_count \rangle$ must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int162int8` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5
Cambricon@155chb
__mlu_entry__ void kernel(int8_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int8_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_short2char(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int8_t), NRAM2GDRAM);
}
```

4.37 __bang_short2int

```
void __bang_short2int(int32_t *dst,
                      int16_t *src,
                      int src_count,
                      int dst_position,
                      int src_position)
```

This function converts type of $\langle src \rangle$ from `int16_t` to `int32_t` element-wisely in round-to-zero mode and saves the result in $\langle dst \rangle$.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.

- [in] `src_count`: The number of elements.
- [in] `dst_position`: Scale factor of destination vector, i.e., $<dst> \times 2^{<dst_position>}$.
- [in] `src_position`: Scale factor of source vector, i.e., $<src> \times 2^{<src_position>}$.

Return

- `void`.

Remark

- `<src>` and `<dst>` must point to `__nram__` address space;
- `<src_position>` and `<dst_position>` must be in the range [-127, 127];
- `<src>` cannot be overlapped with `<dst>`;
- `<src_count>` must be greater than zero;
- This function was deprecated from CNCC v4.0.0. Use `__bang_int162int32` instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 322`;
- CNCC Version: `cncc --version >= 3.0.0`;
- Cambricon BANG Compute Arch Version: `cncc -bang-arch >= compute_30`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= tp_322`.

Example

```
#include <bang.h>

#define SIZE 128
#define DST_POS 3
#define SRC_POS 5

__mlu_entry__ void kernel(int32_t *dst, int16_t *src) {
    __nram__ int16_t src_tmp[SIZE];
    __nram__ int32_t dst_tmp[SIZE];
    __memcpy(src_tmp, src, SIZE * sizeof(int16_t), GDRAM2NRAM);
    __bang_short2int(dst_tmp, src_tmp, SIZE, DST_POS, SRC_POS);
    __memcpy(dst, dst_tmp, SIZE * sizeof(int32_t), NRAM2GDRAM);
}
```

4.38 __bang_sub_const

```
void __bang_sub_const(half *dst,
                      half *src,
                      half value,
                      int elem_count)

void __bang_sub_const(float *dst,
                      float *src,
                      float value,
                      int elem_count)
```

This function subtracts `<value>` from `<elem_count>` elements of `<src>` and saves the result in

<dst>. See the table [Floating Point Calculation of Stream and Scalar Binary Operation Functions](#) for accuracy information.

Parameters

- [out] dst: The address of destination vector.
- [in] src: The address of source vector.
- [in] value: The source scalar.
- [in] elem_count: The number of elements in source vector.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <src> and <dst> must point to __nram__ address space;
- <src> can be overlapped with <dst>;
- <elem_count> * sizeof(type) must be divisible by 128 on (m)tp_2xx;
- The address of <src> and <dst> must be 64-byte aligned on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_sub_scalar](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.8.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- See the example of [__bang_add_const](#) for more details.

4.39 __bang_unlock

```
void __bang_unlock(int lock_id_0,
                   int lock_id_1)
```

Releases lock_id.

Parameters

- [in] lock_id_0: The first lock id.
- [in] lock_id_1: The second lock id.

Return

- void.

Remark

- This function was deprecated from CNCC v4.0.0;
- [__bang_unlock](#) is always paired with [__bang_lock](#);
- For MLU Core, lock_id_0 should be equal to lock_id_1;
- For MPU Core, lock_id_0 should be not equal to lock_id_1;
- [__bang_unlock](#) can be used after [__memcpy](#) from to GDRAM, whose size is larger than 64KB.

Instruction Pipeline

- NA.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= mtp_2xx.

Example

- See the example of [__bang_unlock](#) for more detail.

4.40 __memcpy_nram_to_nram

```
void __memcpy_nram_to_nram(void *dst,
                           const void *src,
                           int size,
                           int dst_stride,
                           int src_stride,
                           int segnum)
```

Copies <size> bytes data from <src> source address to <dst> destination address. Both <src> and <dst> are in __nram__ address space.

We recommended this function for memory copy within __nram__ address space on (m)tp_2xx series for better performance, but it has 128-bytes alignment constraint.

Parameters

- [out] dst: The address of destination area.
- [in] src: The address of source area.
- [in] size: The number of bytes to be copied.
- [in] dst_stride: Destination address stride.
- [in] src_stride: Source address stride.
- [in] segnum: The number of data blocks to be copied.

Return

- void.

Remark

- <size> must be greater than zero and divisible by 128;
- <segnum> is in the range [1, 4096];
- <src> and <dst> must point to __nram__ address space;
- The address of <dst> and <src> must be 64-byte aligned on (m)tp_2xx;
- <dst_stride> and <src_stride> must be greater than or equal to zero, and must be divisible by 64;
- This function was deprecated from CNCC v4.0.0. Use [__bang_move](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;

- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx.`

Example

- None.

4.41 __nramset

```
void __nramset(void *dst,
               int elem_count,
               char value)
```

```
void __nramset(void *dst,
               int elem_count,
               short value)
```

```
void __nramset(void *dst,
               int elem_count,
               float value)
```

```
void __nramset(void *dst,
               int elem_count,
               half value)
```

```
void __nramset(void *dst,
               int elem_count,
               int value)
```

Sets a vector in `__nram__` address space pointed by `<dst>` to the specified `<value>`.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `elem_count`: The number of elements to be set.
- [in] `value`: Value to be set.

Return

- `void`.

Remark

- `<dst>` must point to `__nram__` address space;
- `<elem_count>` must be greater than zero;
- The address of `<dst>` must be 64-byte aligned on `(m)tp_2xx`;
- `<elem_count> * sizeof(type)` must be divisible by 64 on `(m)tp_2xx`;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 2.12.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (m)tp_2xx`.

Example

- None.

4.42 __nramset_float

```
void __nramset_float(float *dst,
                      int elem_count,
                      float value)
```

Sets a vector in __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <elem_count> must be greater than zero;
- <dst> must point to __nram__ address space;
- The address of <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(type) must be divisible by 64 on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 3.5.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.43 __nramset_half

```
void __nramset_half(half *dst,
                     int elem_count,
                     half value)
```

Sets a vector in __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <dst> must point to __nram__ address space;
- <elem_count> must be greater than zero;
- The address of <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(type) must be divisible by 64 on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.44 __nramset_int

```
void __nramset_int(int *dst,
                    int elem_count,
                    int value)
```

Sets a vector in __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <dst> must point to __nram__ address space;
- <elem_count> must be greater than zero;
- The address of <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(type) must be divisible by 64 on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: __BANG_ARCH__ >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.45 __nramset_short

```
void __nramset_short(short *dst,
                      int elem_count,
                      short value)
```

Sets a vector in __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <dst> must point to __nram__ address space;
- <elem_count> must be greater than zero;
- The address of <dst> must be 64-byte aligned on (m)tp_2xx;
- <elem_count> * sizeof(type) must be divisible by 64 on (m)tp_2xx;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: --BANG_ARCH-- >= 200;
- CNCC Version: cncc --version >= 2.12.0;
- Cambricon BANG Compute Arch Version: cncc --bang-arch >= compute_20;
- MLU Compute Arch Version: cncc --bang-mlu-arch >= (m)tp_2xx.

Example

- None.

4.46 __nramset_unsigned_int

```
void __nramset_unsigned_int(unsigned int *dst,
                           int elem_count,
                           unsigned int value)
```

Sets a vector in __nram__ address space pointed by <dst> to the specified <value>.

Parameters

- [out] dst: The address of destination vector.
- [in] elem_count: The number of elements to be set.
- [in] value: Value to be set.

Return

- void.

Remark

- <dst> must point to __nram__ address space;
- <elem_count> must be greater than zero;

- The address of <dst> must be 64-byte aligned on $(\text{m})\text{tp_2xx}$;
- $\langle \text{elem_count} \rangle * \text{sizeof}(\text{type})$ must be divisible by 64 on $(\text{m})\text{tp_2xx}$;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.5.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (\text{m})\text{tp_2xx}`.

Example

- None.

4.47 __nramset_unsigned_short

```
void __nramset_unsigned_short(unsigned short *dst,
                             int elem_count,
                             unsigned short value)
```

Sets a vector in `__nram__` address space pointed by <dst> to the specified <value>.

Parameters

- [out] `dst`: The address of destination vector.
- [in] `elem_count`: The number of elements to be set.
- [in] `value`: Value to be set.

Return

- `void`.

Remark

- <dst> must point to `__nram__` address space;
- <elem_count> must be greater than zero;
- The address of <dst> must be 64-byte aligned on $(\text{m})\text{tp_2xx}$;
- $\langle \text{elem_count} \rangle * \text{sizeof}(\text{type})$ must be divisible by 64 on $(\text{m})\text{tp_2xx}$;
- This function was deprecated from CNCC v4.0.0. Use [__bang_write_value](#) instead.

Instruction Pipeline

- Compute.

Requirements

- Cambricon BANG Version: `__BANG_ARCH__ >= 200`;
- CNCC Version: `cncc --version >= 3.5.0`;
- Cambricon BANG Compute Arch Version: `cncc --bang-arch >= compute_20`;
- MLU Compute Arch Version: `cncc --bang-mlu-arch >= (\text{m})\text{tp_2xx}`.

Example

- None.



5 Appendix

5.1 Rounding Mode

Table 5.1: Rounding Mode

Rounding Mode	Abbreviation	Description	Example of Float2Int
round-to-zero	tz	Round towards zero.	$tz(1.564) = 1$ $tz(-1.564) = -1$
round-off-zero	oz	Round away from zero.	$oz(1.364) = 2$ $oz(-1.364) = -2$
round-up	up	Round towards positive infinity.	$up(1.364) = 2$ $up(-1.564) = -1$
round-down	dn	Round towards negative infinity.	$dn(1.564) = 1$ $dn(-1.364) = -2$
round-nearest-off-zero	rd	Round away from zero when fraction is not less than 0.5. Otherwise, round towards zero.	$rd(1.55) = 2$ $rd(1.35) = 1$ $rd(-2.5) = -3$
round-math	rm	Round towards negative infinity after adding(non-negative) or subtracting(negative) 0.5.	$rm(1.5) = 2$ $rm(-1.5) = -1$
round-nearest-even	rn	Round to nearest even number when fraction is 0.5. Round away from zero when fraction is greater than 0.5. Otherwise, round towards zero.	$rn(-2.5) = -2$ $rn(2.6) = 3$ $rn(1.1) = 1$ $rn(0.5) = 0$
round-stochastic	sr	Stochastic Rounding.	NA

Note:

Rounding modes `rm` and `rn` are only supported on (`m`)tp_3xx and higher.

5.2 Floating Point Calculation

5.2.1 Stream and Scalar Binary Operation Functions

The situation of stream and scalar binary operation functions processing Number/Inf/NaN is as follows:

Table 5.2: Floating Point Calculation of Stream and Scalar Binary Operation Functions

src0	src1	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number	Number/Inf	Number/Inf	Number/Inf
Number	Inf	Sat	Sat/Inf/NaN	Inf/NaN	Sat/Inf/NaN
Number	NaN	Sat	Sat/NaN	NaN	Sat/NaN
Inf	Number	Sat	Sat/Inf/NaN	Inf/NaN	Sat/Inf/NaN
Inf	Inf	Sat	Sat/Inf/NaN	Inf/NaN	Sat/Inf/NaN
Inf	NaN	Sat	Sat/NaN	NaN	Sat/NaN
NaN	Number	Sat	Sat/NaN	NaN	Sat/NaN
NaN	Inf	Sat	Sat/NaN	NaN	Sat/NaN
NaN	NaN	Sat	Sat/NaN	NaN	Sat/NaN

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.2.2 Stream and Scalar Unary Operation Functions

The situation of stream and scalar unary operation functions processing Number/Inf/NaN is as follows:

Table 5.3: Floating Point Calculation of Stream and Scalar Unary Operation Functions

src	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number/Inf/NaN	Number/Inf/NaN	Number/Inf/NaN
Inf	Number	Number/Inf/NaN	Number/Inf/NaN	Number/Inf/NaN
NaN	Sat	Sat/NaN	NaN	Sat/NaN

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.2.3 Stream and Scalar Comparison Functions

The situation of stream and scalar comparison functions processing Number/Inf/NaN is as follows:

Table 5.4: Floating Point Calculation of Stream and Scalar Comparison Functions

src0	src1	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	0/1	0/1	0/1	0/1
Number	Inf	0/1	0/1	0/1	0/1
Number	NaN	0	0	0	0/1
Inf	Number	0/1	0/1	0/1	0/1
Inf	Inf	0/1	0/1	0/1	0/1
Inf	NaN	0	0	0	0/1
NaN	Number	0	0	0	0/1
NaN	Inf	0	0	0	0/1
NaN	NaN	0	0	0	0/1

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.2.4 Element-wise Stream and Scalar Comparison Operation Functions

The situation of stream and scalar comparison functions processing Number/Inf/NaN is as follows:

Table 5.5: Element-wise Floating Point Calculation of Stream
and Scalar Comparison Operation Functions

src0	src1	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number	Number	Number	Number
Number	Inf	Number/Inf	Number/Inf	Number/Inf	Number/Inf
Number	NaN	Sat	Number	Number	Number/Inf
Inf	Number	Number/Inf	Number/Inf	Number/Inf	Number/Inf
Inf	Inf	Inf	Inf	Inf	Inf
Inf	NaN	Sat	Sat/Inf	Inf	Sat/NaN/Inf
NaN	Number	Sat	Sat/NaN	NaN	Sat/NaN
NaN	Inf	Sat	Sat/NaN	NaN	Sat/NaN/Inf
NaN	NaN	Sat	Sat/NaN	NaN	Sat/NaN

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.2.5 Non-element-wise Stream Comparison Operation Functions

The situation of stream comparison functions processing Number/Inf/NaN is as follows:

Table 5.6: Non-element-wise Floating Point Calculation of Stream Comparison Operation Functions

src	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number	Number	Number
Inf	Inf	Inf	Inf	Inf
NaN	NaN	NaN	NaN	NaN
Number/Inf	Number/Inf	Number/Inf	Number/Inf	Number/Inf
Number/NaN	Number	Number	Number	Number/Inf
NaN/Number	NaN	NaN	NaN	Number/Inf
Inf/NaN	Inf	Inf	Inf	Inf/NaN
NaN/Inf	NaN	NaN	NaN	Inf/NaN
Number/Inf,NaN	Number/Inf	Number/Inf	Number/Inf	Number/Inf/NaN
NaN/Number,Inf	NaN	NaN	NaN	Number/Inf/NaN

Note:

- Number means finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.
- Number/Inf ,NaN means that the first operand is Number, and the other operands are Inf and NaN.
- NaN/Number ,Inf means that the first operand is NaN, and the other operands are Number and Inf.

5.2.6 Stream and Scalar Binary Logic and Bit Operation Functions

The situation of stream and scalar binary logic and bit operation functions processing Number/Inf/NaN is as follows:

Table 5.7: Floating Point Calculation of Stream and Scalar Binary Logic and Bit Operation Functions

src0	src1	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	0/1	0/1	0/1	0/1
Number	Inf	0/1	0/1	0/1	0/1
Number	NaN	0/1	0/1	0/1	0/1
Inf	Number	0/1	0/1	0/1	0/1
Inf	Inf	1	1	1	1
Inf	NaN	1	1	1	1
NaN	Number	0/1	0/1	0/1	0/1
NaN	Inf	1	1	1	1
NaN	NaN	1	1	1	1

Note:

- Number means finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.2.7 Stream and Scalar Unary Logic and Bit Operation Functions

The situation of stream and scalar unary logic and bit operation functions processing Number/Inf/NaN is as follows:

Table 5.8: Floating Point Calculation of Stream and Scalar Unary Logic and Bit Operation Functions

src0	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	0/1	0/1	0/1	0/1
Inf	0	0	0	0
NaN	0	0	0	0

Note:

- Number means finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.2.8 Stream and Scalar Type Conversion from Floating Point to Integer

The situation of stream and scalar type conversion from floating point to integer processing Number/Inf/NaN is as follows:

Table 5.9: Floating Point Calculation of Stream and Scalar Type
Conversion from Floating Point to Integer

src0	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number	Number	Number
+Inf	+Sat	+Sat	+Sat	+Sat
-Inf	-Sat	-Sat	-Sat	-Sat
Nan	Sat	Sat	Sat	Sat/0

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- +Sat means positive saturation number.
- -Sat means negative saturation number.
- +Inf means positive infinity number.
- -Inf means negative infinity number.
- NaN means Not a Number.

5.2.9 Stream and Scalar Type Conversion from Floating Point to Floating Point

The situation of stream and scalar type conversion from floating point to floating point processing Number/Inf/NaN is as follows:

Table 5.10: Floating Point Calculation of Stream and Scalar Type Conversion from Floating Point to Floating Point

src0	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number/Inf	Number/Inf	Number/Inf
+Inf	+Sat	+Sat/+Inf	+Inf	+Sat/+Inf
-Inf	-Sat	-Sat/-Inf	-Inf	-Sat/-Inf
NaN	Sat	Sat/NaN	NaN	Sat/NaN

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- +Sat means positive saturation number.
- -Sat means negative saturation number.
- Inf means infinity number, including positive infinity and negative infinity.
- +Inf means positive infinity number.
- -Inf means negative infinity number.
- NaN means Not a Number.

5.2.10 Stream and Scalar Type Conversion from Integer to Floating Point

The situation of stream and scalar type conversion from integer to floating point processing Number is as follows:

Table 5.11: Floating Point Calculation of Stream and Scalar Type Conversion from Integer to Floating Point

src0	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number/Inf	Number/Inf	Number/Inf

Note:

- Number means finite number.
- Inf means infinity number, including positive infinity and negative infinity.

5.2.11 Stream and Scalar Div Operation Function

The situation of stream and scalar div operation function processing Number/Inf/NaN is as follows:

Table 5.12: Floating Point Calculation of Stream and Scalar Div Operation Function

src0	src1	(m)tp_1xx (m)tp_2xx	tp_322	mtp_372	mtp_592
Number	Number	Number	Number/Inf/NaN	Number/NaN	Number/Inf/NaN
Number	Inf	Number	Number	NaN	Number
Number	NaN	Sat	Sat/NaN	NaN	NaN
Inf	Number	Sat	Sat/Inf	Inf/NaN	Sat/Inf
Inf	Inf	Sat	Sat/NaN	NaN	NaN
Inf	NaN	Sat	Sat/NaN	NaN	NaN
NaN	Number	Sat	Sat/NaN	NaN	NaN
NaN	Inf	Sat	Sat/NaN	NaN	NaN
NaN	NaN	Sat	Sat/NaN	NaN	NaN

Note:

- Number means finite number.
- Sat means positive or negative saturation number, which belongs to finite number.
- Inf means infinity number, including positive infinity and negative infinity.
- NaN means Not a Number.

5.3 Mathematical Functions

This section specifies all the functions of the C/C++ standard library mathematical functions that are supported in device-side code.

Table 5.13: Mathematical Functions and Macros

Function	Description
abs	Returns the absolute value of the integer argument.
fabsf	Returns the absolute value of the floating-point argument.
acosf	Returns the arc cosine of the floating-point argument.
cosf	Returns the cosine of the floating-point argument.

continues on next page

Table 5.13 – continued from previous page

Function	Description
coshf	Returns the hyperbolic cosine of the floating-point argument.
acoshf	Returns the inverse hyperbolic cosine of the floating-point argument.
sinf	Returns the sine of the floating-point argument.
sinhf	Returns the hyperbolic sine of the floating-point argument.
asinf	Returns the arc sine of the floating-point argument.
asinhf	Returns the inverse hyperbolic sine of the floating-point argument.
tanf	Returns the tangent of the floating-point argument.
atanf	Returns the arc tangent of the floating-point argument.
atanhf	Returns the inverse hyperbolic tangent of the floating-point argument.
atan2f	Returns the arc tangent of two floating-point arguments.
ceilf	Returns the smallest integral that is greater than or equal to the floating-point argument.
floor	Returns the largest integral that is less than or equal to the argument.
copysignf	Returns a value whose magnitude is taken from first argument and whose sign is taken from second argument.
expf	Returns e raised to the power of the floating-point argument.
expm1f	Returns e raised to the power of the floating-point argument minus 1.
exp2f	Returns 2 raised to the power of the floating-point argument.
fmaxf	Returns the maximum of two floating-point arguments.
fminf	Returns the minimum of two floating-point arguments.
isnan	Returns a nonzero value if the argument is NaN.
isinf	Returns a nonzero value if the argument is INF.
log2f	Returns the base-2 logarithm of the floating-point argument.
log10f	Returns the base-10 logarithm of the floating-point argument.
log1pf	Returns the logarithm of 1 plus the floating-point argument.
logbf	Returns the exponent of the floating-point argument.
powf	Returns the power of the floating-point argument.

continues on next page

Table 5.13 – continued from previous page

Function	Description
roundf	Returns the round to nearest integer away from zero of the floating-point argument.
sqrtf	Returns the square root of the floating-point argument.
scalbnf	Returns the multiplication of the first argument by 2 to the power of the second argument.
truncf	Returns the rounded integer value of argument in floating format.

Cambricon@155chb