

ReproBLAS

Generated by Doxygen 1.8.10

Mon Jan 25 2016 21:01:29

Contents

1	File Index	1
1.1	File List	1
2	File Documentation	3
2.1	include/idxd.h File Reference	3
2.1.1	Detailed Description	10
2.1.2	Macro Definition Documentation	11
2.1.2.1	DIWIDTH	11
2.1.2.2	idxd_DICAPACITY	11
2.1.2.3	idxd_DIENDURANCE	11
2.1.2.4	idxd_DIMAXFOLD	12
2.1.2.5	idxd_DIMAXINDEX	12
2.1.2.6	idxd_DMCOMPRESSION	12
2.1.2.7	idxd_DMEXPANSION	12
2.1.2.8	idxd_SICAPACITY	13
2.1.2.9	idxd_SIENDURANCE	13
2.1.2.10	idxd_SIMAXFOLD	13
2.1.2.11	idxd_SIMAXINDEX	13
2.1.2.12	idxd_SMCOMPRESSION	14
2.1.2.13	idxd_SMEXPANSION	14
2.1.2.14	SIWIDTH	14
2.1.3	Typedef Documentation	14
2.1.3.1	double_complex_indexed	14
2.1.3.2	double_indexed	15
2.1.3.3	float_complex_indexed	15
2.1.3.4	float_indexed	15
2.1.4	Function Documentation	15
2.1.4.1	idxd_cciconv_sub(const int fold, const float_complex_indexed *X, void *conv) . .	15
2.1.4.2	idxd_ccmconv_sub(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, void *conv)	16
2.1.4.3	idxd_cialloc(const int fold)	17
2.1.4.4	idxd_cicadd(const int fold, const void *X, float_complex_indexed *Y)	17

2.1.4.5	<code>idxd_cicconv(const int fold, const void *X, float_complex_indexed *Y)</code>	18
2.1.4.6	<code>idxd_cicdeposit(const int fold, const void *X, float_complex_indexed *Y)</code>	19
2.1.4.7	<code>idxd_ciciadd(const int fold, const float_complex_indexed *X, float_complex_indexed *Y)</code>	19
2.1.4.8	<code>idxd_ciciaddv(const int fold, const int N, const float_complex_indexed *X, const int incX, float_complex_indexed *Y, const int incY)</code>	20
2.1.4.9	<code>idxd_ciciset(const int fold, const float_complex_indexed *X, float_complex_indexed *Y)</code>	20
2.1.4.10	<code>idxd_cicupdate(const int fold, const void *X, float_complex_indexed *Y)</code>	21
2.1.4.11	<code>idxd_cinegate(const int fold, float_complex_indexed *X)</code>	21
2.1.4.12	<code>idxd_cinum(const int fold)</code>	21
2.1.4.13	<code>idxd_ciprint(const int fold, const float_complex_indexed *X)</code>	22
2.1.4.14	<code>idxd_cirenorm(const int fold, float_complex_indexed *X)</code>	23
2.1.4.15	<code>idxd_cisetzero(const int fold, float_complex_indexed *X)</code>	23
2.1.4.16	<code>idxd_cisiset(const int fold, const float_indexed *X, float_complex_indexed *Y)</code>	23
2.1.4.17	<code>idxd_cisize(const int fold)</code>	24
2.1.4.18	<code>idxd_cisupdate(const int fold, const float X, float_complex_indexed *Y)</code>	24
2.1.4.19	<code>idxd_cmcadd(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	25
2.1.4.20	<code>idxd_cmccconv(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	26
2.1.4.21	<code>idxd_cmcdeposit(const int fold, const void *X, float *priY, const int incpriY)</code>	26
2.1.4.22	<code>idxd_cmcadd(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	27
2.1.4.23	<code>idxd_cmcset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	27
2.1.4.24	<code>idxd_cmcupdate(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	28
2.1.4.25	<code>idxd_cmddenorm(const int fold, const float *priX)</code>	28
2.1.4.26	<code>idxd_cmnegate(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	29
2.1.4.27	<code>idxd_cmprint(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	29
2.1.4.28	<code>idxd_cmrenorm(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	30
2.1.4.29	<code>idxd_cmsetzero(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	30
2.1.4.30	<code>idxd_cmsmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	30
2.1.4.31	<code>idxd_cmsrescale(const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	31
2.1.4.32	<code>idxd_cmsupdate(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	31
2.1.4.33	<code>idxd_ddiconv(const int fold, const double_indexed *X)</code>	32

2.1.4.34	<code>idxd_ddmconv(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	32
2.1.4.35	<code>idxd_dialloc(const int fold)</code>	33
2.1.4.36	<code>idxd_dibound(const int fold, const int N, const double X, const double S)</code>	33
2.1.4.37	<code>idxd_didadd(const int fold, const double X, double_indexed *Y)</code>	33
2.1.4.38	<code>idxd_didconv(const int fold, const double X, double_indexed *Y)</code>	34
2.1.4.39	<code>idxd_diddeposit(const int fold, const double X, double_indexed *Y)</code>	34
2.1.4.40	<code>idxd_didiadd(const int fold, const double_indexed *X, double_indexed *Y)</code>	35
2.1.4.41	<code>idxd_didiaddsq(const int fold, const double scaleX, const double_indexed *X, const double scaleY, double_indexed *Y)</code>	35
2.1.4.42	<code>idxd_didiadv(const int fold, const int N, const double_indexed *X, const int incX, double_indexed *Y, const int incY)</code>	36
2.1.4.43	<code>idxd_didiset(const int fold, const double_indexed *X, double_indexed *Y)</code>	36
2.1.4.44	<code>idxd_didupdate(const int fold, const double X, double_indexed *Y)</code>	36
2.1.4.45	<code>idxd_dindex(const double X)</code>	37
2.1.4.46	<code>idxd_dinegate(const int fold, double_indexed *X)</code>	37
2.1.4.47	<code>idxd_dinum(const int fold)</code>	37
2.1.4.48	<code>idxd_diprint(const int fold, const double_indexed *X)</code>	38
2.1.4.49	<code>idxd_direnorm(const int fold, double_indexed *X)</code>	38
2.1.4.50	<code>idxd_disetzero(const int fold, double_indexed *X)</code>	38
2.1.4.51	<code>idxd_dsize(const int fold)</code>	39
2.1.4.52	<code>idxd_dmbins(const int X)</code>	39
2.1.4.53	<code>idxd_dmdadd(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	40
2.1.4.54	<code>idxd_dmdconv(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	41
2.1.4.55	<code>idxd_dmddeposit(const int fold, const double X, double *priY, const int incpriY)</code>	41
2.1.4.56	<code>idxd_dmdenorm(const int fold, const double *priX)</code>	42
2.1.4.57	<code>idxd_dmdmadd(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	42
2.1.4.58	<code>idxd_dmdmaddsq(const int fold, const double scaleX, const double *priX, const int incpriX, const double *carX, const int inccarX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	43
2.1.4.59	<code>idxd_dmdmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	43
2.1.4.60	<code>idxd_dmdrescale(const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	44
2.1.4.61	<code>idxd_dmdupdate(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	44
2.1.4.62	<code>idxd_dmindex(const double *priX)</code>	45
2.1.4.63	<code>idxd_dmindex0(const double *priX)</code>	45

2.1.4.64	<code>idxd_dmnegate(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	46
2.1.4.65	<code>idxd_dmprint(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	46
2.1.4.66	<code>idxd_dmrenorm(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	46
2.1.4.67	<code>idxd_dmsetzero(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	47
2.1.4.68	<code>idxd_dscale(const double X)</code>	47
2.1.4.69	<code>idxd_sialloc(const int fold)</code>	48
2.1.4.70	<code>idxd_sibound(const int fold, const int N, const float X, const float S)</code>	48
2.1.4.71	<code>idxd_sindex(const float X)</code>	49
2.1.4.72	<code>idxd_sinegate(const int fold, float_indexed *X)</code>	49
2.1.4.73	<code>idxd_sinum(const int fold)</code>	49
2.1.4.74	<code>idxd_siprint(const int fold, const float_indexed *X)</code>	50
2.1.4.75	<code>idxd_sirenorm(const int fold, float_indexed *X)</code>	50
2.1.4.76	<code>idxd_sisadd(const int fold, const float X, float_indexed *Y)</code>	50
2.1.4.77	<code>idxd_sisconv(const int fold, const float X, float_indexed *Y)</code>	51
2.1.4.78	<code>idxd_sisdeposit(const int fold, const float X, float_indexed *Y)</code>	51
2.1.4.79	<code>idxd_sisetzzero(const int fold, float_indexed *X)</code>	52
2.1.4.80	<code>idxd_sisiadd(const int fold, const float_indexed *X, float_indexed *Y)</code>	52
2.1.4.81	<code>idxd_sisiaddsq(const int fold, const float scaleX, const float_indexed *X, const float scaleY, float_indexed *Y)</code>	52
2.1.4.82	<code>idxd_sisiadv(const int fold, const int N, const float_indexed *X, const int incX, float_indexed *Y, const int incY)</code>	53
2.1.4.83	<code>idxd_sisiset(const int fold, const float_indexed *X, float_indexed *Y)</code>	53
2.1.4.84	<code>idxd_sisize(const int fold)</code>	54
2.1.4.85	<code>idxd_sisupdate(const int fold, const float X, float_indexed *Y)</code>	54
2.1.4.86	<code>idxd_smbins(const int X)</code>	54
2.1.4.87	<code>idxd_smdenorm(const int fold, const float *priX)</code>	55
2.1.4.88	<code>idxd_smindex(const float *priX)</code>	55
2.1.4.89	<code>idxd_smindex0(const float *priX)</code>	56
2.1.4.90	<code>idxd_smnegate(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	56
2.1.4.91	<code>idxd_smprint(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	56
2.1.4.92	<code>idxd_smrenorm(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	57
2.1.4.93	<code>idxd_smsadd(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	57
2.1.4.94	<code>idxd_smsconv(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	58
2.1.4.95	<code>idxd_smsdeposit(const int fold, const float X, float *priY, const int incpriY)</code>	59

2.1.4.96	<code>idxd_smsetzero(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	59
2.1.4.97	<code>idxd_smsmadd(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	60
2.1.4.98	<code>idxd_smsmaddsq(const int fold, const float scaleX, const float *priX, const int incpriX, const float *carX, const int inccarX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	60
2.1.4.99	<code>idxd_smsmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	61
2.1.4.100	<code>idxd_smsrescale(const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	61
2.1.4.101	<code>idxd_smsupdate(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	62
2.1.4.102	<code>idxd_ssacle(const float X)</code>	62
2.1.4.103	<code>idxd_ssiconv(const int fold, const float_indexed *X)</code>	63
2.1.4.104	<code>idxd_ssmconv(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	63
2.1.4.105	<code>idxd_uftp(const double X)</code>	64
2.1.4.106	<code>idxd_uftp(const float X)</code>	64
2.1.4.107	<code>idxd_zialloc(const int fold)</code>	64
2.1.4.108	<code>idxd_zidiset(const int fold, const double_indexed *X, double_complex_indexed *Y)</code>	65
2.1.4.109	<code>idxd_zidupdate(const int fold, const double X, double_complex_indexed *Y)</code>	65
2.1.4.110	<code>idxd_zinegate(const int fold, double_complex_indexed *X)</code>	66
2.1.4.111	<code>idxd_zinum(const int fold)</code>	67
2.1.4.112	<code>idxd_ziprint(const int fold, const double_complex_indexed *X)</code>	67
2.1.4.113	<code>idxd_zirenorm(const int fold, double_complex_indexed *X)</code>	67
2.1.4.114	<code>idxd_zisetzero(const int fold, double_complex_indexed *X)</code>	68
2.1.4.115	<code>idxd_zisize(const int fold)</code>	68
2.1.4.116	<code>idxd_zizadd(const int fold, const void *X, double_complex_indexed *Y)</code>	68
2.1.4.117	<code>idxd_zizconv(const int fold, const void *X, double_complex_indexed *Y)</code>	69
2.1.4.118	<code>idxd_zizdeposit(const int fold, const void *X, double_complex_indexed *Y)</code>	69
2.1.4.119	<code>idxd_ziziadd(const int fold, const double_complex_indexed *X, double_complex_indexed *Y)</code>	70
2.1.4.120	<code>idxd_ziziaddv(const int fold, const int N, const double_complex_indexed *X, const int incX, double_complex_indexed *Y, const int incY)</code>	70
2.1.4.121	<code>idxd_ziziset(const int fold, const double_complex_indexed *X, double_complex_indexed *Y)</code>	71
2.1.4.122	<code>idxd_zizupdate(const int fold, const void *X, double_complex_indexed *Y)</code>	72
2.1.4.123	<code>idxd_zmdenorm(const int fold, const double *priX)</code>	72
2.1.4.124	<code>idxd_zmdmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	73

2.1.4.125	<code>idxd_zmdrescale(const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	73
2.1.4.126	<code>idxd_zmdupdate(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	74
2.1.4.127	<code>idxd_zmnegate(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	75
2.1.4.128	<code>idxd_zmprint(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	75
2.1.4.129	<code>idxd_zmrenorm(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	76
2.1.4.130	<code>idxd_zmsetzero(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	76
2.1.4.131	<code>idxd_zmzadd(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	76
2.1.4.132	<code>idxd_zmzconv(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	77
2.1.4.133	<code>idxd_zmzdeposit(const int fold, const void *X, double *priY, const int incpriY)</code>	77
2.1.4.134	<code>idxd_zmzmadd(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	78
2.1.4.135	<code>idxd_zmzmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	78
2.1.4.136	<code>idxd_zmzupdate(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	79
2.1.4.137	<code>idxd_zziconv_sub(const int fold, const double_complex_indexed *X, void *conv)</code>	79
2.1.4.138	<code>idxd_zzmconv_sub(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, void *conv)</code>	80
2.2	<code>include/idxdBLAS.h</code> File Reference	80
2.2.1	Detailed Description	84
2.2.2	Function Documentation	84
2.2.2.1	<code>idxdBLAS_camax_sub(const int N, const void *X, const int incX, void *amax)</code>	84
2.2.2.2	<code>idxdBLAS_camaxm_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)</code>	85
2.2.2.3	<code>idxdBLAS_cicdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float_indexed *Z)</code>	85
2.2.2.4	<code>idxdBLAS_cicdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float_indexed *Z)</code>	86
2.2.2.5	<code>idxdBLAS_cicgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, float_complex_indexed *C, const int ldc)</code>	86
2.2.2.6	<code>idxdBLAS_cicgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, float_complex_indexed *Y, const int incY)</code>	87
2.2.2.7	<code>idxdBLAS_cicsum(const int fold, const int N, const void *X, const int incX, float_indexed *Y)</code>	88

2.2.2.8	<code>idxdBLAS_cmcdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	88
2.2.2.9	<code>idxdBLAS_cmcdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	89
2.2.2.10	<code>idxdBLAS_cmcsu(m(const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	89
2.2.2.11	<code>idxdBLAS_damax(const int N, const double *X, const int incX)</code>	90
2.2.2.12	<code>idxdBLAS_damaxm(const int N, const double *X, const int incX, const double *Y, const int incY)</code>	90
2.2.2.13	<code>idxdBLAS_didasum(const int fold, const int N, const double *X, const int incX, double_indexed *Y)</code>	91
2.2.2.14	<code>idxdBLAS_diddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double_indexed *Z)</code>	91
2.2.2.15	<code>idxdBLAS_didgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, double_indexed *C, const int ldc)</code>	92
2.2.2.16	<code>idxdBLAS_didgemv(const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, double_indexed *Y, const int incY)</code>	93
2.2.2.17	<code>idxdBLAS_didssq(const int fold, const int N, const double *X, const int incX, const double scaleY, double_indexed *Y)</code>	93
2.2.2.18	<code>idxdBLAS_didsum(const int fold, const int N, const double *X, const int incX, double_indexed *Y)</code>	94
2.2.2.19	<code>idxdBLAS_dizasum(const int fold, const int N, const void *X, const int incX, double_indexed *Y)</code>	94
2.2.2.20	<code>idxdBLAS_dizssq(const int fold, const int N, const void *X, const int incX, const double scaleY, double_indexed *Y)</code>	94
2.2.2.21	<code>idxdBLAS_dmdasum(const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	95
2.2.2.22	<code>idxdBLAS_dmddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	95
2.2.2.23	<code>idxdBLAS_dmdssq(const int fold, const int N, const double *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	96
2.2.2.24	<code>idxdBLAS_dmdsum(const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	96
2.2.2.25	<code>idxdBLAS_dmzasum(const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	97
2.2.2.26	<code>idxdBLAS_dmzssq(const int fold, const int N, const void *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	97
2.2.2.27	<code>idxdBLAS_samax(const int N, const float *X, const int incX)</code>	98
2.2.2.28	<code>idxdBLAS_samaxm(const int N, const float *X, const int incX, const float *Y, const int incY)</code>	98
2.2.2.29	<code>idxdBLAS_sicasum(const int fold, const int N, const void *X, const int incX, float_indexed *Y)</code>	99

2.2.2.30	<code>idxdBLAS_sicssq(const int fold, const int N, const void *X, const int incX, const float scaleY, float_indexed *Y)</code>	99
2.2.2.31	<code>idxdBLAS_sisasum(const int fold, const int N, const float *X, const int incX, float_indexed *Y)</code>	100
2.2.2.32	<code>idxdBLAS_sisdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float_indexed *Z)</code>	100
2.2.2.33	<code>idxdBLAS_sisgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, float_indexed *C, const int ldc)</code>	101
2.2.2.34	<code>idxdBLAS_sisgemv(const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, float_indexed *Y, const int incY)</code>	102
2.2.2.35	<code>idxdBLAS_sisssq(const int fold, const int N, const float *X, const int incX, const float scaleY, float_indexed *Y)</code>	102
2.2.2.36	<code>idxdBLAS_sissum(const int fold, const int N, const float *X, const int incX, float_indexed *Y)</code>	103
2.2.2.37	<code>idxdBLAS_smcasum(const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	103
2.2.2.38	<code>idxdBLAS_smcssq(const int fold, const int N, const void *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	104
2.2.2.39	<code>idxdBLAS_smsasum(const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	104
2.2.2.40	<code>idxdBLAS_smsdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	105
2.2.2.41	<code>idxdBLAS_smsssq(const int fold, const int N, const float *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	105
2.2.2.42	<code>idxdBLAS_smssum(const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	106
2.2.2.43	<code>idxdBLAS_zamax_sub(const int N, const void *X, const int incX, void *amax)</code>	106
2.2.2.44	<code>idxdBLAS_zamaxm_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)</code>	107
2.2.2.45	<code>idxdBLAS_zizdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_indexed *Z)</code>	107
2.2.2.46	<code>idxdBLAS_zizdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_indexed *Z)</code>	108
2.2.2.47	<code>idxdBLAS_zizgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, double_complex_indexed *C, const int ldc)</code>	108
2.2.2.48	<code>idxdBLAS_zizgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, double_complex_indexed *Y, const int incY)</code>	109
2.2.2.49	<code>idxdBLAS_zizsum(const int fold, const int N, const void *X, const int incX, double_indexed *Y)</code>	110
2.2.2.50	<code>idxdBLAS_zmzdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	110

2.2.2.51	<code>idxdBLAS_zmzdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	111
2.2.2.52	<code>idxdBLAS_zmzsum(const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	111
2.3	<code>include/reproBLAS.h</code> File Reference	112
2.3.1	Detailed Description	115
2.3.2	Function Documentation	116
2.3.2.1	<code>reproBLAS_cdotc_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)</code>	116
2.3.2.2	<code>reproBLAS_cdotu_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)</code>	116
2.3.2.3	<code>reproBLAS_cgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)</code>	117
2.3.2.4	<code>reproBLAS_cgmv(const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)</code>	117
2.3.2.5	<code>reproBLAS_csum_sub(const int N, const void *X, const int incX, void *sum)</code>	118
2.3.2.6	<code>reproBLAS_dasum(const int N, const double *X, const int incX)</code>	118
2.3.2.7	<code>reproBLAS_ddot(const int N, const double *X, const int incX, const double *Y, const int incY)</code>	119
2.3.2.8	<code>reproBLAS_dgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)</code>	119
2.3.2.9	<code>reproBLAS_dgmv(const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)</code>	121
2.3.2.10	<code>reproBLAS_dnrm2(const int N, const double *X, const int incX)</code>	122
2.3.2.11	<code>reproBLAS_dsum(const int N, const double *X, const int incX)</code>	122
2.3.2.12	<code>reproBLAS_dzasum(const int N, const void *X, const int incX)</code>	123
2.3.2.13	<code>reproBLAS_dznrm2(const int N, const void *X, int incX)</code>	123
2.3.2.14	<code>reproBLAS_rcdotc_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)</code>	124
2.3.2.15	<code>reproBLAS_rcdotu_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)</code>	124
2.3.2.16	<code>reproBLAS_rcgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)</code>	125
2.3.2.17	<code>reproBLAS_rcgmv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)</code>	125
2.3.2.18	<code>reproBLAS_rcsum_sub(const int fold, const int N, const void *X, const int incX, void *sum)</code>	127
2.3.2.19	<code>reproBLAS_rdasum(const int fold, const int N, const double *X, const int incX)</code>	127

2.3.2.20	reproBLAS_rddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY)	128
2.3.2.21	reproBLAS_rdgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)	128
2.3.2.22	reproBLAS_rdgemv(const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)	129
2.3.2.23	reproBLAS_rdnrm2(const int fold, const int N, const double *X, const int incX)	130
2.3.2.24	reproBLAS_rddsum(const int fold, const int N, const double *X, const int incX)	130
2.3.2.25	reproBLAS_rdzasum(const int fold, const int N, const void *X, const int incX)	131
2.3.2.26	reproBLAS_rdznm2(const int fold, const int N, const void *X, int incX)	131
2.3.2.27	reproBLAS_rsasum(const int fold, const int N, const float *X, const int incX)	132
2.3.2.28	reproBLAS_rscasum(const int fold, const int N, const void *X, const int incX)	132
2.3.2.29	reproBLAS_rscnrm2(const int fold, const int N, const void *X, const int incX)	133
2.3.2.30	reproBLAS_rsdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY)	133
2.3.2.31	reproBLAS_rsgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)	134
2.3.2.32	reproBLAS_rsgemv(const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)	135
2.3.2.33	reproBLAS_rsnrm2(const int fold, const int N, const float *X, const int incX)	136
2.3.2.34	reproBLAS_rssum(const int fold, const int N, const float *X, const int incX)	136
2.3.2.35	reproBLAS_rzdotc_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)	136
2.3.2.36	reproBLAS_rzdotu_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)	137
2.3.2.37	reproBLAS_rzgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)	137
2.3.2.38	reproBLAS_rzgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)	139
2.3.2.39	reproBLAS_rzsum_sub(const int fold, const int N, const void *X, int incX, void *sum)	140
2.3.2.40	reproBLAS_sasum(const int N, const float *X, const int incX)	140
2.3.2.41	reproBLAS_scasum(const int N, const void *X, const int incX)	141
2.3.2.42	reproBLAS_scnrm2(const int N, const void *X, const int incX)	141
2.3.2.43	reproBLAS_sdot(const int N, const float *X, const int incX, const float *Y, const int incY)	142
2.3.2.44	reproBLAS_sgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)	142

2.3.2.45	reproBLAS_sgemv(const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)	143
2.3.2.46	reproBLAS_snrm2(const int N, const float *X, const int incX)	144
2.3.2.47	reproBLAS_ssum(const int N, const float *X, const int incX)	144
2.3.2.48	reproBLAS_zdotc_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)	145
2.3.2.49	reproBLAS_zdotu_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)	145
2.3.2.50	reproBLAS_zgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc) . . .	146
2.3.2.51	reproBLAS_zgemv(const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)	146
2.3.2.52	reproBLAS_zsum_sub(const int N, const void *X, int incX, void *sum)	148

Index**149**

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

include/ idxd.h	
idxd.h defines the indexed types and the lower level functions associated with their use	3
include/ idxdBLAS.h	
idxdBLAS.h defines BLAS Methods that operate on indexed types	80
include/ idxdMPI.h	??
include/ reproBLAS.h	
ReproBLAS.h defines reproducible BLAS Methods	112

Chapter 2

File Documentation

2.1 include/idxd.h File Reference

[idxd.h](#) defines the indexed types and the lower level functions associated with their use.

```
#include <stddef.h>
#include <stdlib.h>
#include <float.h>
```

Macros

- `#define DIWIDTH 40`
Indexed double precision bin width.
- `#define SIWIDTH 13`
Indexed single precision bin width.
- `#define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)`
Indexed double precision maximum index.
- `#define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)`
Indexed single precision maximum index.
- `#define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)`
The maximum double precision fold supported by the library.
- `#define idxd_SIMAXFOLD (idxd_SIMAXINDEX + 1)`
The maximum single precision fold supported by the library.
- `#define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))`
Indexed double precision deposit endurance.
- `#define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))`
Indexed single precision deposit endurance.
- `#define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))`
Indexed double precision capacity.
- `#define idxd_SICAPACITY (idxd_SIENDURANCE*(1.0/FLT_EPSILON - 1.0))`
Indexed single precision capacity.
- `#define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`
Indexed double precision compression factor.
- `#define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`
Indexed single precision compression factor.
- `#define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`
Indexed double precision expansion factor.
- `#define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`
Indexed single precision expansion factor.

Typedefs

- typedef double [double_indexed](#)
The indexed double datatype.
- typedef double [double_complex_indexed](#)
The indexed complex double datatype.
- typedef float [float_indexed](#)
The indexed float datatype.
- typedef float [float_complex_indexed](#)
The indexed complex float datatype.

Functions

- size_t [idxd_disize](#) (const int fold)
indexed double precision size
- size_t [idxd_zisize](#) (const int fold)
indexed complex double precision size
- size_t [idxd_sisize](#) (const int fold)
indexed single precision size
- size_t [idxd_cisize](#) (const int fold)
indexed complex single precision size
- [double_indexed](#) * [idxd_dialloc](#) (const int fold)
indexed double precision allocation
- [double_complex_indexed](#) * [idxd_zialloc](#) (const int fold)
indexed complex double precision allocation
- [float_indexed](#) * [idxd_sialloc](#) (const int fold)
indexed single precision allocation
- [float_complex_indexed](#) * [idxd_cialloc](#) (const int fold)
indexed complex single precision allocation
- int [idxd_dinum](#) (const int fold)
indexed double precision size
- int [idxd_zinum](#) (const int fold)
indexed complex double precision size
- int [idxd_sinum](#) (const int fold)
indexed single precision size
- int [idxd_cinum](#) (const int fold)
indexed complex single precision size
- double [idxd_dibound](#) (const int fold, const int N, const double X, const double S)
Get indexed double precision summation error bound.
- float [idxd_sibound](#) (const int fold, const int N, const float X, const float S)
Get indexed single precision summation error bound.
- const double * [idxd_dmbins](#) (const int X)
Get indexed double precision reference bins.
- const float * [idxd_smbins](#) (const int X)
Get indexed single precision reference bins.
- int [idxd_dindex](#) (const double X)
Get index of double precision.
- int [idxd_dmindex](#) (const double *priX)
Get index of manually specified indexed double precision.
- int [idxd_dmindex0](#) (const double *priX)
Check if index of manually specified indexed double precision is 0.

- int `idxd_sindex` (const float X)
Get index of single precision.
- int `idxd_smindex` (const float *priX)
Get index of manually specified indexed single precision.
- int `idxd_smindex0` (const float *priX)
Check if index of manually specified indexed single precision is 0.
- int `idxd_dmdenorm` (const int fold, const double *priX)
Check if indexed type has denormal bits.
- int `idxd_zmdenorm` (const int fold, const double *priX)
Check if indexed type has denormal bits.
- int `idxd_smdenorm` (const int fold, const float *priX)
Check if indexed type has denormal bits.
- int `idxd_cmdenorm` (const int fold, const float *priX)
Check if indexed type has denormal bits.
- void `idxd_diprint` (const int fold, const `double_indexed` *X)
Print indexed double precision.
- void `idxd_dmprint` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)
Print manually specified indexed double precision.
- void `idxd_ziprint` (const int fold, const `double_complex_indexed` *X)
Print indexed complex double precision.
- void `idxd_zmprint` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)
Print manually specified indexed complex double precision.
- void `idxd_siprint` (const int fold, const `float_indexed` *X)
Print indexed single precision.
- void `idxd_smprint` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)
Print manually specified indexed single precision.
- void `idxd_ciprint` (const int fold, const `float_complex_indexed` *X)
Print indexed complex single precision.
- void `idxd_cmprint` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)
Print manually specified indexed complex single precision.
- void `idxd_didiset` (const int fold, const `double_indexed` *X, `double_indexed` *Y)
Set indexed double precision ($Y = X$)
- void `idxd_dmdmset` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
Set manually specified indexed double precision ($Y = X$)
- void `idxd_ziziset` (const int fold, const `double_complex_indexed` *X, `double_complex_indexed` *Y)
Set indexed complex double precision ($Y = X$)
- void `idxd_zmzmset` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
Set manually specified indexed complex double precision ($Y = X$)
- void `idxd_zidiset` (const int fold, const `double_indexed` *X, `double_complex_indexed` *Y)
Set indexed complex double precision to indexed double precision ($Y = X$)
- void `idxd_zmdmset` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
Set manually specified indexed complex double precision to manually specified indexed double precision ($Y = X$)
- void `idxd_sisiset` (const int fold, const `float_indexed` *X, `float_indexed` *Y)
Set indexed single precision ($Y = X$)
- void `idxd_smsmset` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
Set manually specified indexed single precision ($Y = X$)
- void `idxd_ciciset` (const int fold, const `float_complex_indexed` *X, `float_complex_indexed` *Y)

- Set indexed complex single precision ($Y = X$)*
 - void `idxd_cmcmset` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
- Set manually specified indexed complex single precision ($Y = X$)*
 - void `idxd_cisaset` (const int fold, const `float_indexed` *X, `float_complex_indexed` *Y)
- Set indexed complex single precision to indexed single precision ($Y = X$)*
 - void `idxd_cmsmset` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
- Set manually specified indexed complex single precision to manually specified indexed single precision ($Y = X$)*
 - void `idxd_disetzero` (const int fold, `double_indexed` *X)
- Set indexed double precision to 0 ($X = 0$)*
 - void `idxd_dmsetzero` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
- Set manually specified indexed double precision to 0 ($X = 0$)*
 - void `idxd_zisetzero` (const int fold, `double_complex_indexed` *X)
- Set indexed double precision to 0 ($X = 0$)*
 - void `idxd_zmsetzero` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
- Set manually specified indexed complex double precision to 0 ($X = 0$)*
 - void `idxd_sisetzero` (const int fold, `float_indexed` *X)
- Set indexed single precision to 0 ($X = 0$)*
 - void `idxd_smsetzero` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
- Set manually specified indexed single precision to 0 ($X = 0$)*
 - void `idxd_cisetzero` (const int fold, `float_complex_indexed` *X)
- Set indexed single precision to 0 ($X = 0$)*
 - void `idxd_cmsetzero` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
- Set manually specified indexed complex single precision to 0 ($X = 0$)*
 - void `idxd_didiadd` (const int fold, const `double_indexed` *X, `double_indexed` *Y)
- Add indexed double precision ($Y += X$)*
 - void `idxd_dmdmadd` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
- Add manually specified indexed double precision ($Y += X$)*
 - void `idxd_ziziadd` (const int fold, const `double_complex_indexed` *X, `double_complex_indexed` *Y)
- Add indexed complex double precision ($Y += X$)*
 - void `idxd_zmzmadd` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
- Add manually specified indexed complex double precision ($Y += X$)*
 - void `idxd_sisiadd` (const int fold, const `float_indexed` *X, `float_indexed` *Y)
- Add indexed single precision ($Y += X$)*
 - void `idxd_smsmadd` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
- Add manually specified indexed single precision ($Y += X$)*
 - void `idxd_ciciadd` (const int fold, const `float_complex_indexed` *X, `float_complex_indexed` *Y)
- Add indexed complex single precision ($Y += X$)*
 - void `idxd_cmcmadd` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
- Add manually specified indexed complex single precision ($Y += X$)*
 - void `idxd_didiaddv` (const int fold, const int N, const `double_indexed` *X, const int incX, `double_indexed` *Y, const int incY)
- Add indexed double precision vectors ($Y += X$)*
 - void `idxd_ziziaddv` (const int fold, const int N, const `double_complex_indexed` *X, const int incX, `double_complex_indexed` *Y, const int incY)
- Add indexed complex double precision vectors ($Y += X$)*
 - void `idxd_zmzmaddv` (const int fold, const int N, const `double_indexed` *X, const int incX, `double_indexed` *Y, const int incY)

- void `idxd_sisiaddv` (const int fold, const int N, const `float_indexed` *X, const int incX, `float_indexed` *Y, const int incY)
Add indexed single precision vectors (Y += X)
- void `idxd_ciciaddv` (const int fold, const int N, const `float_complex_indexed` *X, const int incX, `float_complex_indexed` *Y, const int incY)
Add indexed complex single precision vectors (Y += X)
- void `idxd_didadd` (const int fold, const double X, `double_indexed` *Y)
Add double precision to indexed double precision (Y += X)
- void `idxd_dmdadd` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
Add double precision to manually specified indexed double precision (Y += X)
- void `idxd_zizadd` (const int fold, const void *X, `double_complex_indexed` *Y)
Add complex double precision to indexed complex double precision (Y += X)
- void `idxd_zmzadd` (const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)
Add complex double precision to manually specified indexed complex double precision (Y += X)
- void `idxd_sisadd` (const int fold, const float X, `float_indexed` *Y)
Add single precision to indexed single precision (Y += X)
- void `idxd_smsadd` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
Add single precision to manually specified indexed single precision (Y += X)
- void `idxd_cicadd` (const int fold, const void *X, `float_complex_indexed` *Y)
Add complex single precision to indexed complex single precision (Y += X)
- void `idxd_cmcadd` (const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)
Add complex single precision to manually specified indexed complex single precision (Y += X)
- void `idxd_didupdate` (const int fold, const double X, `double_indexed` *Y)
Update indexed double precision with double precision (X -> Y)
- void `idxd_dmdupdate` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
Update manually specified indexed double precision with double precision (X -> Y)
- void `idxd_zizupdate` (const int fold, const void *X, `double_complex_indexed` *Y)
Update indexed complex double precision with complex double precision (X -> Y)
- void `idxd_zmzupdate` (const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)
Update manually specified indexed complex double precision with complex double precision (X -> Y)
- void `idxd_zidupdate` (const int fold, const double X, `double_complex_indexed` *Y)
Update indexed complex double precision with double precision (X -> Y)
- void `idxd_zmdupdate` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
Update manually specified indexed complex double precision with double precision (X -> Y)
- void `idxd_sisupdate` (const int fold, const float X, `float_indexed` *Y)
Update indexed single precision with single precision (X -> Y)
- void `idxd_smsupdate` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
Update manually specified indexed single precision with single precision (X -> Y)
- void `idxd_cicupdate` (const int fold, const void *X, `float_complex_indexed` *Y)
Update indexed complex single precision with complex single precision (X -> Y)
- void `idxd_cmcupdate` (const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)
Update manually specified indexed complex single precision with complex single precision (X -> Y)
- void `idxd_cisupdate` (const int fold, const float X, `float_complex_indexed` *Y)
Update indexed complex single precision with single precision (X -> Y)
- void `idxd_cmsupdate` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
Update manually specified indexed complex single precision with single precision (X -> Y)
- void `idxd_diddeposit` (const int fold, const double X, `double_indexed` *Y)

- Add double precision to suitably indexed indexed double precision ($Y += X$)*
 - void `idxd_dmddeposit` (const int fold, const double X, double *priY, const int incpriY)
- Add double precision to suitably indexed manually specified indexed double precision ($Y += X$)*
 - void `idxd_zizdeposit` (const int fold, const void *X, `double_complex_indexed` *Y)
- Add complex double precision to suitably indexed indexed complex double precision ($Y += X$)*
 - void `idxd_zmzdeposit` (const int fold, const void *X, double *priY, const int incpriY)
- Add complex double precision to suitably indexed manually specified indexed complex double precision ($Y += X$)*
 - void `idxd_sisdeposit` (const int fold, const float X, `float_indexed` *Y)
- Add single precision to suitably indexed indexed single precision ($Y += X$)*
 - void `idxd_smsdeposit` (const int fold, const float X, float *priY, const int incpriY)
- Add single precision to suitably indexed manually specified indexed single precision ($Y += X$)*
 - void `idxd_cicdeposit` (const int fold, const void *X, `float_complex_indexed` *Y)
- Add complex single precision to suitably indexed indexed complex single precision ($Y += X$)*
 - void `idxd_cmcddeposit` (const int fold, const void *X, float *priY, const int incpriY)
- Add complex single precision to suitably indexed manually specified indexed complex single precision ($Y += X$)*
 - void `idxd_direnorm` (const int fold, `double_indexed` *X)
- Renormalize indexed double precision.*
 - void `idxd_dmrenorm` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
- Renormalize manually specified indexed double precision.*
 - void `idxd_zirenorm` (const int fold, `double_complex_indexed` *X)
- Renormalize indexed complex double precision.*
 - void `idxd_zmrenorm` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
- Renormalize manually specified indexed complex double precision.*
 - void `idxd_sirenorm` (const int fold, `float_indexed` *X)
- Renormalize indexed single precision.*
 - void `idxd_smrenorm` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
- Renormalize manually specified indexed single precision.*
 - void `idxd_cirenorm` (const int fold, `float_complex_indexed` *X)
- Renormalize indexed complex single precision.*
 - void `idxd_cmrenorm` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
- Renormalize manually specified indexed complex single precision.*
 - void `idxd_didconv` (const int fold, const double X, `double_indexed` *Y)
- Convert double precision to indexed double precision ($X \rightarrow Y$)*
 - void `idxd_dmdconv` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
- Convert double precision to manually specified indexed double precision ($X \rightarrow Y$)*
 - void `idxd_zizconv` (const int fold, const void *X, `double_complex_indexed` *Y)
- Convert complex double precision to indexed complex double precision ($X \rightarrow Y$)*
 - void `idxd_zmzconv` (const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)
- Convert complex double precision to manually specified indexed complex double precision ($X \rightarrow Y$)*
 - void `idxd_sisconv` (const int fold, const float X, `float_indexed` *Y)
- Convert single precision to indexed single precision ($X \rightarrow Y$)*
 - void `idxd_smsconv` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
- Convert single precision to manually specified indexed single precision ($X \rightarrow Y$)*
 - void `idxd_cicconv` (const int fold, const void *X, `float_complex_indexed` *Y)
- Convert complex single precision to indexed complex single precision ($X \rightarrow Y$)*
 - void `idxd_cmccconv` (const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)
- Convert complex single precision to manually specified indexed complex single precision ($X \rightarrow Y$)*
 - double `idxd_ddiconv` (const int fold, const `double_indexed` *X)
- Convert indexed double precision to double precision ($X \rightarrow Y$)*

- double [idxd_ddmconv](#) (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)
Convert manually specified indexed double precision to double precision ($X \rightarrow Y$)
- void [idxd_zziconv_sub](#) (const int fold, const [double_complex_indexed](#) *X, void *conv)
Convert indexed complex double precision to complex double precision ($X \rightarrow Y$)
- void [idxd_zzmconv_sub](#) (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, void *conv)
Convert manually specified indexed complex double precision to complex double precision ($X \rightarrow Y$)
- float [idxd_ssiconv](#) (const int fold, const [float_indexed](#) *X)
Convert indexed single precision to single precision ($X \rightarrow Y$)
- float [idxd_ssmconv](#) (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)
Convert manually specified indexed single precision to single precision ($X \rightarrow Y$)
- void [idxd_cciconv_sub](#) (const int fold, const [float_complex_indexed](#) *X, void *conv)
Convert indexed complex single precision to complex single precision ($X \rightarrow Y$)
- void [idxd_ccmconv_sub](#) (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, void *conv)
Convert manually specified indexed complex single precision to complex single precision ($X \rightarrow Y$)
- void [idxd_dinegate](#) (const int fold, [double_indexed](#) *X)
Negate indexed double precision ($X = -X$)
- void [idxd_dmnegate](#) (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
Negate manually specified indexed double precision ($X = -X$)
- void [idxd_zinegate](#) (const int fold, [double_complex_indexed](#) *X)
Negate indexed complex double precision ($X = -X$)
- void [idxd_zmnegate](#) (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
Negate manually specified indexed complex double precision ($X = -X$)
- void [idxd_sinegate](#) (const int fold, [float_indexed](#) *X)
Negate indexed single precision ($X = -X$)
- void [idxd_smnegate](#) (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
Negate manually specified indexed single precision ($X = -X$)
- void [idxd_cinegate](#) (const int fold, [float_complex_indexed](#) *X)
Negate indexed complex single precision ($X = -X$)
- void [idxd_cmnegate](#) (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
Negate manually specified indexed complex single precision ($X = -X$)
- double [idxd_dscale](#) (const double X)
Get a reproducible double precision scale.
- float [idxd_sscale](#) (const float X)
Get a reproducible single precision scale.
- void [idxd_dmdrescale](#) (const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
rescale manually specified indexed double precision sum of squares
- void [idxd_zmdrescale](#) (const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
rescale manually specified indexed complex double precision sum of squares
- void [idxd_smsrescale](#) (const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)
rescale manually specified indexed single precision sum of squares
- void [idxd_cmsrescale](#) (const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)
rescale manually specified indexed complex single precision sum of squares
- double [idxd_dmdmaddsq](#) (const int fold, const double scaleX, const double *priX, const int incpriX, const double *carX, const int inccarX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)

Add manually specified indexed double precision scaled sums of squares ($Y += X$)

- double `idxd_didiaddsq` (const int fold, const double scaleX, const `double_indexed` *X, const double scaleY, `double_indexed` *Y)

Add indexed double precision scaled sums of squares ($Y += X$)

- float `idxd_smsmaddsq` (const int fold, const float scaleX, const float *priX, const int incpriX, const float *carX, const int inccarX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)

Add manually specified indexed single precision scaled sums of squares ($Y += X$)

- float `idxd_sisiaddsq` (const int fold, const float scaleX, const `float_indexed` *X, const float scaleY, `float_indexed` *Y)

Add indexed single precision scaled sums of squares ($Y += X$)

- double `idxd_ufp` (const double X)

unit in the first place

- float `idxd_ufpf` (const float X)

unit in the first place

2.1.1 Detailed Description

`idxd.h` defines the indexed types and the lower level functions associated with their use.

This header is modeled after `cblas.h`, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- di - indexed double (`double_indexed`)
- zi - indexed complex double (`double_complex_indexed`)
- si - indexed float (`float_indexed`)
- ci - indexed complex float (`float_complex_indexed`)
- dm - manually specified indexed double (`double, double`)
- zm - manually specified indexed complex double (`double, double`)
- sm - manually specified indexed float (`float, float`)
- cm - manually specified indexed complex float (`float, float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Indexed types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in `config.h`. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use. Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

2.1.2 Macro Definition Documentation

2.1.2.1 `#define DIWIDTH 40`

Indexed double precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.2 `#define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))`

Indexed double precision capacity.

The maximum number of double precision numbers that can be summed using indexed double precision. Applies also to indexed complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.3 `#define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))`

Indexed double precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to indexed complex double precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.4 `#define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)`

The maximum double precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.5 `#define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)`

Indexed double precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.6 `#define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`

Indexed double precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.7 `#define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`

Indexed double precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.8 #define idxd_SICAPACITY (idxd_SIENDURANCE*(1.0/FLT_EPSILON - 1.0))

Indexed single precision capacity.

The maximum number of single precision numbers that can be summed using indexed single precision. Applies also to indexed complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.9 #define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))

Indexed single precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to indexed complex single precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.10 #define idxd_SIMAXFOLD (idxd_SIMAXINDEX + 1)

The maximum single precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.11 #define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)

Indexed single precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.12 `#define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`

Indexed single precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.13 `#define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`

Indexed single precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.14 `#define SIWIDTH 13`

Indexed single precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.3 Typedef Documentation

2.1.3.1 `typedef double double_complex_indexed`

The indexed complex double datatype.

To allocate a [double_complex_indexed](#), call [idxd_zialloc\(\)](#)

Warning

A [double_complex_indexed](#) is, under the hood, an array of `double`. Therefore, if you have defined an array of [double_complex_indexed](#), you must index it by multiplying the index into the array by the number of underlying `double` that make up the [double_complex_indexed](#). This number can be obtained by a call to [idxd_zinum\(\)](#)

2.1.3.2 typedef double double_indexed

The indexed double datatype.

To allocate a `double_indexed`, call `idxd_dialloc()`

Warning

A `double_indexed` is, under the hood, an array of `double`. Therefore, if you have defined an array of `double_indexed`, you must index it by multiplying the index into the array by the number of underlying `double` that make up the `double_indexed`. This number can be obtained by a call to `idxd_dinum()`

2.1.3.3 typedef float float_complex_indexed

The indexed complex float datatype.

To allocate a `float_complex_indexed`, call `idxd_cialloc()`

Warning

A `float_complex_indexed` is, under the hood, an array of `float`. Therefore, if you have defined an array of `float_complex_indexed`, you must index it by multiplying the index into the array by the number of underlying `float` that make up the `float_complex_indexed`. This number can be obtained by a call to `idxd_cinum()`

2.1.3.4 typedef float float_indexed

The indexed float datatype.

To allocate a `float_indexed`, call `idxd_sialloc()`

Warning

A `float_indexed` is, under the hood, an array of `float`. Therefore, if you have defined an array of `float_indexed`, you must index it by multiplying the index into the array by the number of underlying `float` that make up the `float_indexed`. This number can be obtained by a call to `idxd_sinum()`

2.1.4 Function Documentation

2.1.4.1 void idxd_cciconv_sub (const int fold, const float_complex_indexed * X, void * conv)

Convert indexed complex single precision to complex single precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.2 void idxd_ccmconv_sub (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*,
void * *conv*)

Convert manually specified indexed complex single precision to complex single precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.3 `float_complex_indexed* idxd_cialloc (const int fold)`

indexed complex single precision allocation

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

a freshly allocated indexed type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.4 `void idxd_cicadd (const int fold, const void * X, float_complex_indexed * Y)`

Add complex single precision to indexed complex single precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.5 void `idxd_cicconv` (const int *fold*, const void * *X*, float_complex_indexed * *Y*)

Convert complex single precision to indexed complex single precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.6 void idxd_cicdeposit (const int *fold*, const void * *X*, float_complex_indexed * *Y*)

Add complex single precision to suitably indexed indexed complex single precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_cicupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_cicdeposit\(\)](#) to deposit a maximum of [idxd_SIENDURANCE](#) elements into Y before renormalizing Y with [idxd_cirenorm\(\)](#). After any number of successive calls of [idxd_cicdeposit\(\)](#) on Y, you must renormalize Y with [idxd_cirenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.7 void idxd_ciciadd (const int *fold*, const float_complex_indexed * *X*, float_complex_indexed * *Y*)

Add indexed complex single precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.8 `void idxd_ciciaddv (const int fold, const int N, const float_complex_indexed * X, const int incX, float_complex_indexed * Y, const int incY)`

Add indexed complex single precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.9 `void idxd_ciciset (const int fold, const float_complex_indexed * X, float_complex_indexed * Y)`

Set indexed complex single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.10 void idxd_cicupdate (const int *fold*, const void * *X*, float_complex_indexed * *Y*)

Update indexed complex single precision with complex single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of *X* respectively.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.11 void idxd_cinegate (const int *fold*, float_complex_indexed * *X*)

Negate indexed complex single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.12 int idxd_cinum (const int *fold*)

indexed complex single precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in `float`) of the indexed type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.13 `void idxd_ciprint (const int fold, const float_complex_indexed * X)`

Print indexed complex single precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.14 void idxd_cirenorm (const int *fold*, float_complex_indexed * *X*)

Renormalize indexed complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.15 void idxd_cisetzzero (const int *fold*, float_complex_indexed * *X*)

Set indexed single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.16 void idxd_cisisset (const int *fold*, const float_indexed * *X*, float_complex_indexed * *Y*)

Set indexed complex single precision to indexed single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.17 `size_t idxd_cisize (const int fold)`

indexed complex single precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.18 `void idxd_cisupdate (const int fold, const float X, float_complex_indexed * Y)`

Update indexed complex single precision with single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.19 void idxd_cmcadd (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add complex single precision to manually specified indexed complex single precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.20 void idxd_cmconv (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Convert complex single precision to manually specified indexed complex single precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.21 void idxd_cmcdposit (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*)

Add complex single precision to suitably indexed manually specified indexed complex single precision (Y += X)

Performs the operation Y += X on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_cmupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_cmcdposit\(\)](#) to deposit a maximum of [idxd_SIENDURANCE](#) elements into Y before renormalizing Y with [idxd_cmrenorm\(\)](#). After any number of successive calls of [idxd_cmcdposit\(\)](#) on Y, you must renormalize Y with [idxd_cmrenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.22 void idxd_cmcmadd (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add manually specified indexed complex single precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.23 void idxd_cmcmset (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Set manually specified indexed complex single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.24 void idxd_cmupdate (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Update manually specified indexed complex single precision with complex single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.25 int idxd_cmdenorm (const int *fold*, const float * *priX*)

Check if indexed type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

>0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.26 void idxd_cmnegate (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Negate manually specified indexed complex single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.27 void idxd_cmprint (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*)

Print manually specified indexed complex single precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.28 void idxd_cmrenorm (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified indexed complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.29 void idxd_cmsetzero (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Set manually specified indexed complex single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.30 void idxd_cmsmset (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Set manually specified indexed complex single precision to manually specified indexed single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.31 void idxd_cmsrescale (const int *fold*, const float *X*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

rescale manually specified indexed complex single precision sum of squares

Rescale an indexed complex single precision sum of squares *Y* to *Y'* such that $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$ and `#idxd_sminindex(Y) == idxd_sindex(1.0)`

Note that *Y* is assumed to have an index at least the index of 1.0, and that $X \geq \text{scaleY}$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ($X == \text{\#idxd_sscale}(Y)$ for some float <i>Y</i>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{\#idxd_sscale}(Y)$ for some float <i>Y</i>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector (<code>#idxd_sminindex(Y) >= idxd_sindex(1.0)</code>)
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.32 void idxd_cmsupdate (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Update manually specified indexed complex single precision with single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.33 `double idxd_ddiconv (const int fold, const double_indexed * X)`

Convert indexed double precision to double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.34 `double idxd_ddmconv (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX)`

Convert manually specified indexed double precision to double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Returns

scalar Y

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.35 double_indexed* idxd_dialloc (const int *fold*)

indexed double precision allocation

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returnsa freshly allocated indexed type. (free with `free()`)**Author**

Peter Ahrens

Date

27 Apr 2015

2.1.4.36 double idxd_dibound (const int *fold*, const int *N*, const double *X*, const double *S*)

Get indexed double precision summation error bound.

This is a bound on the absolute error of a summation using indexed types

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	the number of double precision floating point summands
<i>X</i>	the summand of maximum absolute value
<i>S</i>	the value of the sum computed using indexed types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.37 void idxd_didadd (const int *fold*, const double *X*, double_indexed * *Y*)Add double precision to indexed double precision ($Y += X$)Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.38 `void idxd_didconv (const int fold, const double X, double_indexed * Y)`

Convert double precision to indexed double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.39 `void idxd_diddeposit (const int fold, const double X, double_indexed * Y)`

Add double precision to suitably indexed indexed double precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_didupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_diddeposit\(\)](#) to deposit a maximum of [idxd_DIENDURANCE](#) elements into Y before renormalizing Y with [idxd_direnorm\(\)](#). After any number of successive calls of [idxd_diddeposit\(\)](#) on Y, you must renormalize Y with [idxd_direnorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X

<i>Y</i>	indexed scalar <i>Y</i>
----------	-------------------------

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.40 `void idxd_didiadd (const int fold, const double_indexed * X, double_indexed * Y)`

Add indexed double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.41 `double idxd_didiadds (const int fold, const double scaleX, const double_indexed * X, const double scaleY, double_indexed * Y)`

Add indexed double precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where *X* and *Y* represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of <i>X</i> ($scaleX == \#idxd_dscale(Z)$ for some double <i>Z</i>)
<i>X</i>	indexed scalar <i>X</i>
<i>scaleY</i>	scale of <i>Y</i> ($scaleY == \#idxd_dscale(Z)$ for some double <i>Z</i>)
<i>Y</i>	indexed scalar <i>Y</i>

Returns

updated scale of *Y*

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.42 void idxd_didiaddv (const int *fold*, const int *N*, const double_indexed * *X*, const int *incX*, double_indexed * *Y*, const int *incY*)

Add indexed double precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.43 void idxd_didiset (const int *fold*, const double_indexed * *X*, double_indexed * *Y*)

Set indexed double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.44 void idxd_didupdate (const int *fold*, const double *X*, double_indexed * *Y*)

Update indexed double precision with double precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.45 int idxd_dindex (const double *X*)

Get index of double precision.

The index of a non-indexed type is the smallest index an indexed type would need to have to sum it reproducibly. Higher indices correspond to smaller bins.

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.46 void idxd_dinegate (const int *fold*, double_indexed * *X*)

Negate indexed double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.47 int idxd_dinum (const int *fold*)

indexed double precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in `double`) of the indexed type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.48 `void idxd_diprint (const int fold, const double_indexed * X)`

Print indexed double precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.49 `void idxd_direnorm (const int fold, double_indexed * X)`

Renormalize indexed double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.50 `void idxd_disetzero (const int fold, double_indexed * X)`

Set indexed double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.51 `size_t idxd_disize (const int fold)`

indexed double precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.52 `const double* idxd_dmbins (const int X)`

Get indexed double precision reference bins.

returns a pointer to the bins corresponding to the given index

Parameters

<i>X</i>	index
----------	-------

Returns

pointer to constant double precision bins of index *X*

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.53 `void idxd_dmdadd (const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY)`

Add double precision to manually specified indexed double precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.54 `void idxd_dmdconv (const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY)`

Convert double precision to manually specified indexed double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

30 Apr 2015

2.1.4.55 `void idxd_dmddeposit (const int fold, const double X, double * priY, const int incpriY)`

Add double precision to suitably indexed manually specified indexed double precision (Y += X)

Performs the operation Y += X on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_dmdupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_dmddeposit\(\)](#) to deposit a maximum of [idxd_DIENDURANCE](#) elements into Y before renormalizing Y with [idxd_dmrenorm\(\)](#). After any number of successive calls of [idxd_dmddeposit\(\)](#) on Y, you must renormalize Y with [idxd_dmrenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.56 `int idxd_dmdenorm (const int fold, const double * priX)`

Check if indexed type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the indexed type
<i>priX</i>	X's primary vector

Returns

>0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.57 `void idxd_dmdmadd (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, double * priY, const int incpriY, double * carY, const int inccarY)`

Add manually specified indexed double precision (Y += X)

Performs the operation Y += X

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector

<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.58 `double idxd_dmdmaddsq (const int fold, const double scaleX, const double * priX, const int incpriX, const double * carX, const int inccarX, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY)`

Add manually specified indexed double precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of X ($scaleX == \#idxd_dscale(Z)$ for some <code>double Z</code>)
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>scaleY</i>	scale of Y ($scaleY == \#idxd_dscale(Z)$ for some <code>double Z</code>)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Returns

updated scale of Y

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.59 `void idxd_dmdmset (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, double * priY, const int incpriY, double * carY, const int inccarY)`

Set manually specified indexed double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.60 `void idxd_dmdrescale (const int fold, const double X, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY)`

rescale manually specified indexed double precision sum of squares

Rescale an indexed double precision sum of squares Y to Y' such that $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$

Note that Y is assumed to have an index at least the index of 1.0, and that $X \geq \text{scaleY}$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new <i>scaleY</i> ($X == \text{\#idxd_dscale}(Y)$ for some <code>double Y</code>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current <i>scaleY</i> ($\text{scaleY} == \text{\#idxd_dscale}(Y)$ for some <code>double Y</code>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector ($\text{\#idxd_dminindex}(Y) \geq \text{idxd_dindex}(1.0)$)
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.61 `void idxd_dmdupdate (const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY)`

Update manually specified indexed double precision with double precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

5 May 2015

2.1.4.62 `int idxd_dminindex (const double * priX)`

Get index of manually specified indexed double precision.

The index of an indexed type is the bin that it corresponds to. Higher indices correspond to smaller bins.

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

23 Sep 2015

2.1.4.63 `int idxd_dminindex0 (const double * priX)`

Check if index of manually specified indexed double precision is 0.

A quick check to determine if the index is 0

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

>0 if x has index 0, 0 otherwise.

Author

Peter Ahrens

Date

19 May 2015

2.1.4.64 `void idxd_dmnegate (const int fold, double * priX, const int incpriX, double * carX, const int inccarX)`

Negate manually specified indexed double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.65 `void idxd_dmprint (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX)`

Print manually specified indexed double precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.66 `void idxd_dmrenorm (const int fold, double * priX, const int incpriX, double * carX, const int inccarX)`

Renormalize manually specified indexed double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector

<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

23 Sep 2015

2.1.4.67 void idxd_dmsetzero (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Set manually specified indexed double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.68 double idxd_dscale (const double *X*)

Get a reproducible double precision scale.

For any given X, return a reproducible scaling factor Y of the form

$2^{(\text{DIWIDTH} * z)}$ where z is an integer

such that

$Y * 2^{(-\text{DBL_MANT_DIG} - \text{DIWIDTH} - 1)} < X < Y * 2^{(\text{DIWIDTH} + 2)}$

Parameters

<i>X</i>	double precision number to be scaled
----------	--------------------------------------

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.69 `float_indexed* idxd_sialloc (const int fold)`

indexed single precision allocation

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

a freshly allocated indexed type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.70 `float idxd_sibound (const int fold, const int N, const float X, const float S)`

Get indexed single precision summation error bound.

This is a bound on the absolute error of a summation using indexed types

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	the number of single precision floating point summands
<i>X</i>	the summand of maximum absolute value
<i>S</i>	the value of the sum computed using indexed types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

Author

Peter Ahrens
Hong Diep Nguyen

Date

21 May 2015

2.1.4.71 `int idxd_index (const float X)`

Get index of single precision.

The index of a non-indexed type is the smallest index an indexed type would need to have to sum it reproducibly. Higher indicies correspond to smaller bins.

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.72 `void idxd_sinegate (const int fold, float_indexed * X)`

Negate indexed single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.73 `int idxd_sinum (const int fold)`

indexed single precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in `float`) of the indexed type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.74 void idxd_siprint (const int *fold*, const float_indexed * *X*)

Print indexed single precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

AuthorHong Diep Nguyen
Peter Ahrens**Date**

27 Apr 2015

2.1.4.75 void idxd_sirenorm (const int *fold*, float_indexed * *X*)

Renormalize indexed single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

AuthorHong Diep Nguyen
Peter Ahrens**Date**

27 Apr 2015

2.1.4.76 void idxd_sisadd (const int *fold*, const float *X*, float_indexed * *Y*)

Add single precision to indexed single precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.77 void idxd_sisconv (const int *fold*, const float *X*, float_indexed * *Y*)

Convert single precision to indexed single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.78 void idxd_sisdeposit (const int *fold*, const float *X*, float_indexed * *Y*)

Add single precision to suitably indexed indexed single precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_sisupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_sisdeposit\(\)](#) to deposit a maximum of [idxd_SIENDURANCE](#) elements into Y before renormalizing Y with [idxd_sirenorm\(\)](#). After any number of successive calls of [idxd_sisdeposit\(\)](#) on Y, you must renormalize Y with [idxd_sirenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X

Y	indexed scalar Y
---	------------------

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.79 void idxd_sisetzzero (const int *fold*, float_indexed * X)

Set indexed single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
X	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.80 void idxd_sisiadd (const int *fold*, const float_indexed * X, float_indexed * Y)

Add indexed single precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.81 float idxd_sisiaddsqr (const int *fold*, const float *scaleX*, const float_indexed * X, const float *scaleY*, float_indexed * Y)

Add indexed single precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of X ($\text{scaleX} == \text{\#idxd_sscale}(Z)$ for some <code>float Z</code>)
<i>X</i>	indexed scalar X
<i>scaleY</i>	scale of Y ($\text{scaleY} == \text{\#idxd_sscale}(Z)$ for some <code>float Z</code>)
<i>Y</i>	indexed scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.82 `void idxd_sisadv (const int fold, const int N, const float_indexed * X, const int incX, float_indexed * Y, const int incY)`

Add indexed single precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.83 `void idxd_sisaset (const int fold, const float_indexed * X, float_indexed * Y)`

Set indexed single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.84 `size_t idxd_sisize (const int fold)`

indexed single precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.85 `void idxd_sisupdate (const int fold, const float X, float_indexed * Y)`

Update indexed single precision with single precision (*X* -> *Y*)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.86 `const float* idxd_smbins (const int X)`

Get indexed single precision reference bins.

returns a pointer to the bins corresponding to the given index

Parameters

<i>X</i>	index
----------	-------

Returns

pointer to constant single precision bins of index *X*

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.87 `int idxd_smdenorm (const int fold, const float * priX)`

Check if indexed type has denormal bits.

A quick check to determine if calculations involving *X* cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the indexed type
<i>priX</i>	<i>X</i> 's primary vector

Returns

>0 if *x* has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.88 `int idxd_smindex (const float * priX)`

Get index of manually specified indexed single precision.

The index of an indexed type is the bin that it corresponds to. Higher indices correspond to smaller bins.

Parameters

<i>priX</i>	<i>X</i> 's primary vector
-------------	----------------------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

23 Sep 2015

2.1.4.89 `int idxd_smindex0 (const float * priX)`

Check if index of manually specified indexed single precision is 0.

A quick check to determine if the index is 0

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

>0 if x has index 0, 0 otherwise.

Author

Peter Ahrens

Date

19 May 2015

2.1.4.90 `void idxd_smnegate (const int fold, float * priX, const int incpriX, float * carX, const int inccarX)`

Negate manually specified indexed single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.91 `void idxd_smprint (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX)`

Print manually specified indexed single precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector

<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.92 void idxd_smrenorm (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified indexed single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

23 Sep 2015

2.1.4.93 void idxd_smsadd (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add single precision to manually specified indexed single precision (Y += X)

Performs the operation Y += X on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.94 void idxd_smsconv (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Convert single precision to manually specified indexed single precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

30 Apr 2015

2.1.4.95 void idxd_smsdeposit (const int *fold*, const float *X*, float * *priY*, const int *incpriY*)

Add single precision to suitably indexed manually specified indexed single precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_smsupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_smsdeposit\(\)](#) to deposit a maximum of [idxd_SIENDURANCE](#) elements into Y before renormalizing Y with [idxd_smrenorm\(\)](#). After any number of successive calls of [idxd_smsdeposit\(\)](#) on Y, you must renormalize Y with [idxd_smrenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.96 void idxd_smsetzero (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Set manually specified indexed single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.97 void idxd_smsmadd (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add manually specified indexed single precision (Y += X)

Performs the operation Y += X

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.98 float idxd_smsmaddsq (const int *fold*, const float *scaleX*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add manually specified indexed single precision scaled sums of squares (Y += X)

Performs the operation Y += X, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of X ($\text{scaleX} == \text{\#idxd_sscale}(Z)$ for some <code>float Z</code>)
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>scaleY</i>	scale of Y ($\text{scaleY} == \text{\#idxd_sscale}(Z)$ for some <code>double Z</code>)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Returns

updated scale of Y

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.99 `void idxd_smsmset (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX, float * priY, const int incpriY, float * carY, const int inccarY)`

Set manually specified indexed single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.100 `void idxd_smsrescale (const int fold, const float X, const float scaleY, float * priY, const int incpriY, float * carY, const int inccarY)`

rescale manually specified indexed single precision sum of squares

Rescale an indexed single precision sum of squares Y to Y' such that $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$

Note that Y is assumed to have an index at least the index of 1.0, and that $X \geq \text{scaleY}$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ($X == \text{\#idxd_sscale}(Y)$ for some <code>float Y</code>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{\#idxd_sscale}(Y)$ for some <code>float Y</code>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector ($\text{\#idxd_smindex}(Y) \geq \text{idxd_sindex}(1.0)$)
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.101 `void idxd_smsupdate (const int fold, const float X, float * priY, const int incpriY, float * carY, const int inccarY)`

Update manually specified indexed single precision with single precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

5 May 2015

2.1.4.102 `float idxd_sscales (const float X)`

Get a reproducible single precision scale.

For any given X, return a reproducible scaling factor Y of the form

$2^{(\text{SIWIDTH} * z)}$ where z is an integer

such that

$Y * 2^{(-\text{FLT_MANT_DIG} - \text{SIWIDTH} - 1)} < X < Y * 2^{(\text{SIWIDTH} + 2)}$

Parameters

<i>X</i>	single precision number to be scaled
----------	--------------------------------------

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.103 float idxd_ssiconv (const int *fold*, const float_indexed * *X*)

Convert indexed single precision to single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.104 float idxd_ssmconv (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*)

Convert manually specified indexed single precision to single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.105 double idxd_ufp (double *X*)

unit in the first place

This method returns just the implicit 1 in the mantissa of a `double`

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

unit in the first place

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.106 float idxd_ufpf (float *X*)

unit in the first place

This method returns just the implicit 1 in the mantissa of a `float`

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

unit in the first place

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.107 double_complex_indexed* idxd_zialloc (const int *fold*)

indexed complex double precision allocation

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

a freshly allocated indexed type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.108 `void idxd_zidiset (const int fold, const double_indexed * X, double_complex_indexed * Y)`

Set indexed complex double precision to indexed double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.109 `void idxd_zidupdate (const int fold, const double X, double_complex_indexed * Y)`

Update indexed complex double precision with double precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.110 `void idxd_zinegate (const int fold, double_complex_indexed * X)`

Negate indexed complex double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.111 int idxd_zinum (const int *fold*)

indexed complex double precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in `double`) of the indexed type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.112 void idxd_ziprint (const int *fold*, const double_complex_indexed * *X*)

Print indexed complex double precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.113 void idxd_zirenorm (const int *fold*, double_complex_indexed * *X*)

Renormalize indexed complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.114 void `idxd_zisetzero` (const int *fold*, **double_complex_indexed** * *X*)

Set indexed double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.115 size_t `idxd_zisize` (const int *fold*)

indexed complex double precision size

Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

Returns

the size (in bytes) of the indexed type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.116 void `idxd_zizadd` (const int *fold*, const void * *X*, **double_complex_indexed** * *Y*)

Add complex double precision to indexed complex double precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.117 `void idxd_zizconv (const int fold, const void * X, double_complex_indexed * Y)`

Convert complex double precision to indexed complex double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.118 `void idxd_zizdeposit (const int fold, const void * X, double_complex_indexed * Y)`

Add complex double precision to suitably indexed indexed complex double precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_zizupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_zizdeposit\(\)](#) to deposit a maximum of [idxd_DIENDURANCE](#) elements into Y before renormalizing Y with [idxd_zirenorm\(\)](#). After any number of successive calls of [idxd_zizdeposit\(\)](#) on Y, you must renormalize Y with [idxd_zirenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X

Y	indexed scalar Y
---	------------------

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.119 void `idxd_ziziadd` (const int *fold*, const `double_complex_indexed` * *X*, `double_complex_indexed` * *Y*)

Add indexed complex double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.120 void `idxd_ziziaddv` (const int *fold*, const int *N*, const `double_complex_indexed` * *X*, const int *incX*, `double_complex_indexed` * *Y*, const int *incY*)

Add indexed complex double precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.121 void idxd_ziziset (const int *fold*, const double_complex_indexed * *X*, double_complex_indexed * *Y*)

Set indexed complex double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.122 void idxd_zizupdate (const int *fold*, const void * *X*, double_complex_indexed * *Y*)

Update indexed complex double precision with complex double precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of *X* respectively.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.123 int idxd_zmdenorm (const int *fold*, const double * *priX*)

Check if indexed type has denormal bits.

A quick check to determine if calculations involving *X* cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the indexed type
<i>priX</i>	<i>X</i> 's primary vector

Returns

>0 if *x* has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.124 void idxd_zmdmset (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Set manually specified indexed complex double precision to manually specified indexed double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.125 void idxd_zmdrescale (const int *fold*, const double *X*, const double *scaleY*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

rescale manually specified indexed complex double precision sum of squares

Rescale an indexed complex double precision sum of squares Y to Y' such that $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$ and $\text{\#idxd_dmindex}(Y) == \text{\#idxd_dindex}(1.0)$

Note that Y is assumed to have an index at least the the index of 1.0, and that $X \geq \text{scaleY}$

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ($X == \text{\#idxd_dscale}(Y)$ for some double Y) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{\#idxd_dscale}(Y)$ for some double Y) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector ($\text{\#idxd_dmindex}(Y) \geq \text{\#idxd_dindex}(1.0)$)
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.126 `void idxd_zmdupdate (const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY)`

Update manually specified indexed complex double precision with double precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.127 void idxd_zmnegate (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Negate manually specified indexed complex double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.128 void idxd_zmprint (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*)

Print manually specified indexed complex double precision.

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.129 `void idxd_zmrenorm (const int fold, double * priX, const int incpriX, double * carX, const int inccarX)`

Renormalize manually specified indexed complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.130 `void idxd_zmsetzero (const int fold, double * priX, const int incpriX, double * carX, const int inccarX)`

Set manually specified indexed complex double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.131 `void idxd_zmzadd (const int fold, const void * X, double * priY, const int incpriY, double * carY, const int inccarY)`

Add complex double precision to manually specified indexed complex double precision ($Y += X$)

Performs the operation $Y += X$ on an indexed type Y

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.132 void idxd_zmzconv (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Convert complex double precision to manually specified indexed complex double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.133 void idxd_zmzdeposit (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*)

Add complex double precision to suitably indexed manually specified indexed complex double precision (Y += X)

Performs the operation Y += X on an indexed type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd_zmzupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd_zmzdeposit\(\)](#) to deposit a maximum of [idxd_DIENDURANCE](#) elements into Y before renormalizing Y with [idxd_zmrenorm\(\)](#). After any number of successive calls of [idxd_zmzdeposit\(\)](#) on Y, you must renormalize Y with [idxd_zmrenorm\(\)](#) before using any other function on Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.134 void idxd_zmzmadd (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add manually specified indexed complex double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.135 void idxd_zmzmset (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Set manually specified indexed complex double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.136 void idxd_zmzupdate (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Update manually specified indexed complex double precision with complex double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.137 void idxd_zzconv_sub (const int *fold*, const double_complex_indexed * *X*, void * *conv*)

Convert indexed complex double precision to complex double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>X</i>	indexed scalar X
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.138 `void idxd_zzmconv_sub (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, void * conv)`

Convert manually specified indexed complex double precision to complex double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.2 include/idxdBLAS.h File Reference

[idxdBLAS.h](#) defines BLAS Methods that operate on indexed types.

```
#include "idxd.h"
#include "reproBLAS.h"
#include <complex.h>
```

Functions

- float [idxdBLAS_samax](#) (const int N, const float *X, const int incX)
Find maximum absolute value in vector of single precision.
- double [idxdBLAS_damax](#) (const int N, const double *X, const int incX)
Find maximum absolute value in vector of double precision.
- void [idxdBLAS_camax_sub](#) (const int N, const void *X, const int incX, void *amax)
Find maximum magnitude in vector of complex single precision.
- void [idxdBLAS_zamax_sub](#) (const int N, const void *X, const int incX, void *amax)
Find maximum magnitude in vector of complex double precision.

- float `idxdBLAS_samaxm` (const int N, const float *X, const int incX, const float *Y, const int incY)
Find maximum absolute value pairwise product between vectors of single precision.
- double `idxdBLAS_damaxm` (const int N, const double *X, const int incX, const double *Y, const int incY)
Find maximum absolute value pairwise product between vectors of double precision.
- void `idxdBLAS_camaxm_sub` (const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)
Find maximum magnitude pairwise product between vectors of complex single precision.
- void `idxdBLAS_zamaxm_sub` (const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)
Find maximum magnitude pairwise product between vectors of complex double precision.
- void `idxdBLAS_didsum` (const int fold, const int N, const double *X, const int incX, `double_indexed` *Y)
Add to indexed double precision Y the sum of double precision vector X.
- void `idxdBLAS_dmdsum` (const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified indexed double precision Y the sum of double precision vector X.
- void `idxdBLAS_didasum` (const int fold, const int N, const double *X, const int incX, `double_indexed` *Y)
Add to indexed double precision Y the absolute sum of double precision vector X.
- void `idxdBLAS_dmdasum` (const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified indexed double precision Y the absolute sum of double precision vector X.
- double `idxdBLAS_didssq` (const int fold, const int N, const double *X, const int incX, const double scaleY, `double_indexed` *Y)
Add to scaled indexed double precision Y the scaled sum of squares of elements of double precision vector X.
- double `idxdBLAS_dmdssq` (const int fold, const int N, const double *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of double precision vector X.
- void `idxdBLAS_diddot` (const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, `double_indexed` *Z)
Add to indexed double precision Z the dot product of double precision vectors X and Y.
- void `idxdBLAS_dmddot` (const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)
Add to manually specified indexed double precision Z the dot product of double precision vectors X and Y.
- void `idxdBLAS_zizsum` (const int fold, const int N, const void *X, const int incX, `double_indexed` *Y)
Add to indexed complex double precision Y the sum of complex double precision vector X.
- void `idxdBLAS_zmzsum` (const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified indexed complex double precision Y the sum of complex double precision vector X.
- void `idxdBLAS_dizasum` (const int fold, const int N, const void *X, const int incX, `double_indexed` *Y)
Add to indexed double precision Y the absolute sum of complex double precision vector X.
- void `idxdBLAS_dmzasum` (const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified indexed double precision Y the absolute sum of complex double precision vector X.
- double `idxdBLAS_dizssq` (const int fold, const int N, const void *X, const int incX, const double scaleY, `double_indexed` *Y)
Add to scaled indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.
- double `idxdBLAS_dmzssq` (const int fold, const int N, const void *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.
- void `idxdBLAS_zizdotu` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, `double_indexed` *Z)

Add to indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y .

- void `idxdBLAS_zmzdotu` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)

Add to manually specified indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y .

- void `idxdBLAS_zizdotc` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_indexed *Z)

Add to indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y .

- void `idxdBLAS_zmzdotc` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)

Add to manually specified indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y .

- void `idxdBLAS_sisum` (const int fold, const int N, const float *X, const int incX, float_indexed *Y)

Add to indexed single precision Y the sum of single precision vector X .

- void `idxdBLAS_smssum` (const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)

Add to manually specified indexed single precision Y the sum of single precision vector X .

- void `idxdBLAS_sisasum` (const int fold, const int N, const float *X, const int incX, float_indexed *Y)

Add to indexed single precision Y the absolute sum of single precision vector X .

- void `idxdBLAS_smsasum` (const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)

Add to manually specified indexed single precision Y the absolute sum of double precision vector X .

- float `idxdBLAS_sisssq` (const int fold, const int N, const float *X, const int incX, const float scaleY, float_indexed *Y)

Add to scaled indexed single precision Y the scaled sum of squares of elements of single precision vector X .

- float `idxdBLAS_smsssq` (const int fold, const int N, const float *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)

Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of single precision vector X .

- void `idxdBLAS_sisdot` (const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float_indexed *Z)

Add to indexed single precision Z the dot product of single precision vectors X and Y .

- void `idxdBLAS_smsdot` (const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)

Add to manually specified indexed single precision Z the dot product of single precision vectors X and Y .

- void `idxdBLAS_cicsum` (const int fold, const int N, const void *X, const int incX, float_indexed *Y)

Add to indexed complex single precision Y the sum of complex single precision vector X .

- void `idxdBLAS_cmcsun` (const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)

Add to manually specified indexed complex single precision Y the sum of complex single precision vector X .

- void `idxdBLAS_sicasun` (const int fold, const int N, const void *X, const int incX, float_indexed *Y)

Add to indexed single precision Y the absolute sum of complex single precision vector X .

- void `idxdBLAS_smcasun` (const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)

Add to manually specified indexed single precision Y the absolute sum of complex single precision vector X .

- float `idxdBLAS_sicssq` (const int fold, const int N, const void *X, const int incX, const float scaleY, float_indexed *Y)

Add to scaled indexed single precision Y the scaled sum of squares of elements of complex single precision vector X .

- float `idxdBLAS_smcssq` (const int fold, const int N, const void *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)

Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of complex single precision vector X .

- void [idxdBLAS_cicdotu](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, [float_indexed](#) *Z)
Add to indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void [idxdBLAS_cmcdotu](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)
Add to manually specified indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void [idxdBLAS_cicdotc](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, [float_indexed](#) *Z)
Add to indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void [idxdBLAS_cmcdotc](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)
Add to manually specified indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void [idxdBLAS_didgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, [double_indexed](#) *Y, const int incY)
Add to indexed double precision vector Y the matrix-vector product of double precision matrix A and double precision vector X.
- void [idxdBLAS_didgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, [double_indexed](#) *C, const int ldc)
Add to indexed double precision matrix C the matrix-matrix product of double precision matrices A and B.
- void [idxdBLAS_sisgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, [float_indexed](#) *Y, const int incY)
Add to indexed single precision vector Y the matrix-vector product of single precision matrix A and single precision vector X.
- void [idxdBLAS_sisgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, [float_indexed](#) *C, const int ldc)
Add to indexed single precision matrix C the matrix-matrix product of single precision matrices A and B.
- void [idxdBLAS_zizgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, [double_complex_indexed](#) *Y, const int incY)
Add to indexed complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.
- void [idxdBLAS_zizgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, [double_complex_indexed](#) *C, const int ldc)
Add to indexed complex double precision matrix C the matrix-matrix product of complex double precision matrices A and B.
- void [idxdBLAS_cicgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, [float_complex_indexed](#) *Y, const int incY)
Add to indexed complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.
- void [idxdBLAS_cicgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, [float_complex_indexed](#) *C, const int ldc)
Add to indexed complex single precision matrix C the matrix-matrix product of complex single precision matrices A and B.

2.2.1 Detailed Description

[idxdBLAS.h](#) defines BLAS Methods that operate on indexed types.

This header is modeled after `cblas.h`, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- `d` - `double` (`double`)
- `z` - `complex double` (`*void`)
- `s` - `float` (`float`)
- `c` - `complex float` (`*void`)
- `di` - `indexed double` ([double_indexed](#))
- `zi` - `indexed complex double` ([double_complex_indexed](#))
- `si` - `indexed float` ([float_indexed](#))
- `ci` - `indexed complex float` ([float_complex_indexed](#))
- `dm` - `manually specified indexed double` (`double, double`)
- `zm` - `manually specified indexed complex double` (`double, double`)
- `sm` - `manually specified indexed float` (`float, float`)
- `cm` - `manually specified indexed complex float` (`float, float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by `sub`, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Indexed types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many bins are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in `config.h`. If you are unsure of what value to use for , we recommend 3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use. Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

2.2.2 Function Documentation

2.2.2.1 `void idxdBLAS_camax_sub (const int N, const void * X, const int incX, void * amax)`

Find maximum magnitude in vector of complex single precision.

Returns the magnitude of the element of maximum magnitude in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>amax</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.2 void idxdBLAS_camaxm_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *amaxm*)

Find maximum magnitude pairwise product between vectors of complex single precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>amaxm</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.3 void idxdBLAS_cicdotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float_complex_indexed * *Z*)

Add to indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector

<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>indexed</i>	scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.4 void idxdBLAS_cicdotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float_complex_indexed * *Z*)

Add to indexed complex single precision *Z* the unconjugated dot product of complex single precision vectors *X* and *Y*.

Add to *Z* the indexed sum of the pairwise products of *X* and *Y*.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>indexed</i>	scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.5 void idxdBLAS_cicgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, float_complex_indexed * *C*, const int *ldc*)

Add to indexed complex single precision matrix *C* the matrix-matrix product of complex single precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

α is a scalar, *A* and *B* are matrices with $\text{op}(A)$ an *M* by *K* matrix and $\text{op}(B)$ a *K* by *N* matrix, and *C* is an indexed *M* by *N* matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	indexed complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.6 void idxdBLAS_cicgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, float_complex_indexed * *Y*, const int *incY*)

Add to indexed complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha A x + \beta y$ or $y := \alpha A^T x + \beta y$ or $y := \alpha A^H x + \beta y$,

where alpha and beta are scalars, x is a vector, y is an indexed vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha

<i>A</i>	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	indexed complex single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.7 void idxdBLAS_cicsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_complex_indexed * *Y*)

Add to indexed complex single precision Y the sum of complex single precision vector X.

Add to Y the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>indexed</i>	scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.8 void idxdBLAS_cmcdotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float * *manZ*, const int *incmanZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.9 void idxdBLAS_cmcdotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float * *manZ*, const int *incmanZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.10 void idxdBLAS_cmcsu (const int *fold*, const int *N*, const void * *X*, const int *incX*, float * *manY*, const int *incmanY*, float * *carY*, const int *inccarY*)

Add to manually specified indexed complex single precision Y the sum of complex single precision vector X.

Add to Y the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.11 `double idxdBLAS_damax (const int N, const double * X, const int incX)`

Find maximum absolute value in vector of double precision.

Returns the absolute value of the element of maximum absolute value in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute maximum value of X

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.12 `double idxdBLAS_damaxm (const int N, const double * X, const int incX, const double * Y, const int incY)`

Find maximum absolute value pairwise product between vectors of double precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

Parameters

<i>N</i>	vector length
----------	---------------

<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

absolute maximum value multiple of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.13 void idxdBLAS_didasum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double_indexed * *Y*)

Add to indexed double precision Y the absolute sum of double precision vector X.

Add to Y the indexed sum of absolute values of elements in X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.14 void idxdBLAS_diddot (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*, double_indexed * *Z*)

Add to indexed double precision Z the dot product of double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	indexed scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.15 void idxdBLAS_didgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double * *A*, const int *lda*, const double * *B*, const int *ldb*, double_indexed * *C*, const int *ldc*)

Add to indexed double precision matrix *C* the matrix-matrix product of double precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α is a scalar, *A* and *B* are matrices with $\text{op}(A)$ an *M* by *K* matrix and $\text{op}(B)$ a *K* by *N* matrix, and *C* is an indexed *M* by *N* matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose <i>B</i> before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix <i>C</i> .
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix <i>C</i> .
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	double precision matrix of dimension (<i>ma</i> , <i>lda</i>) in row-major or (<i>lda</i> , <i>na</i>) in column-major. (<i>ma</i> , <i>na</i>) is (<i>M</i> , <i>K</i>) if <i>A</i> is not transposed and (<i>K</i> , <i>M</i>) otherwise.
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program. <i>lda</i> must be at least <i>na</i> in row major or <i>ma</i> in column major.
<i>B</i>	double precision matrix of dimension (<i>mb</i> , <i>ldb</i>) in row-major or (<i>ldb</i> , <i>nb</i>) in column-major. (<i>mb</i> , <i>nb</i>) is (<i>K</i> , <i>N</i>) if <i>B</i> is not transposed and (<i>N</i> , <i>K</i>) otherwise.
<i>ldb</i>	the first dimension of <i>B</i> as declared in the calling program. <i>ldb</i> must be at least <i>nb</i> in row major or <i>mb</i> in column major.
<i>C</i>	indexed double precision matrix of dimension (<i>M</i> , <i>ldc</i>) in row-major or (<i>ldc</i> , <i>N</i>) in column-major.
<i>ldc</i>	the first dimension of <i>C</i> as declared in the calling program. <i>ldc</i> must be at least <i>N</i> in row major or <i>M</i> in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.16 void idxdBLAS_didgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const double *alpha*, const double * *A*, const int *lda*, const double * *X*, const int *incX*, double_indexed * *Y*, const int *incY*)

Add to indexed double precision vector *Y* the matrix-vector product of double precision matrix *A* and double precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + y$ or $y := \alpha * A^T * x + y$,

where α is a scalar, x is a vector, y is an indexed vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar α
<i>A</i>	double precision matrix of dimension (<i>M</i> , <i>lda</i>) in row-major or (<i>lda</i> , <i>N</i>) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	double precision vector of at least size <i>N</i> if not transposed or size <i>M</i> otherwise
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed double precision vector <i>Y</i> of at least size <i>M</i> if not transposed or size <i>N</i> otherwise
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.17 double idxdBLAS_didssq (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double *scaleY*, double_indexed * *Y*)

Add to scaled indexed double precision *Y* the scaled sum of squares of elements of double precision vector *X*.

Add to *Y* the scaled indexed sum of the squares of each element of *X*. The scaling of each square is performed using [idxd_dscale\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of <i>Y</i>
<i>Y</i>	indexed scalar <i>Y</i>

Returns

the new scaling factor of *Y*

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.18 void idxdBLAS_didsum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double_indexed * *Y*)

Add to indexed double precision *Y* the sum of double precision vector *X*.

Add to *Y* the indexed sum of *X*.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar <i>Y</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.19 void idxdBLAS_dizasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double_indexed * *Y*)

Add to indexed double precision *Y* the absolute sum of complex double precision vector *X*.

Add to *Y* the indexed sum of magnitudes of elements of *X*.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar <i>Y</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.20 double idxdBLAS_dizssq (const int *fold*, const int *N*, const void * *X*, const int *incX*, const double *scaleY*, double_indexed * *Y*)

Add to scaled indexed double precision *Y* the scaled sum of squares of elements of complex double precision vector *X*.

Add to *Y* the scaled indexed sum of the squares of each element of *X*. The scaling of each square is performed using [idxd_dscale\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>Y</i>	indexed scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.21 void idxdBLAS_dmdasum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double * *manY*, const int *incmanY*, double * *carY*, const int *inccarY*)

Add to manually specified indexed double precision Y the absolute sum of double precision vector X.

Add to Y the indexed sum of absolute values of elements in X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.22 void idxdBLAS_dmddot (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*, double * *manZ*, const int *incmanZ*, double * *carZ*, const int *inccarZ*)

Add to manually specified indexed double precision Z the dot product of double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.23 `double idxdBLAS_dmdssq (const int fold, const int N, const double * X, const int incX, const double scaleY, double * manY, const int incmanY, double * carY, const int inccarY)`

Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of double precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd_dscale\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.24 `void idxdBLAS_dmdsum (const int fold, const int N, const double * X, const int incX, double * manY, const int incmanY, double * carY, const int inccarY)`

Add to manually specified indexed double precision Y the sum of double precision vector X.

Set Y to the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.25 void idxdBLAS_dmzasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double * *manY*, const int *incmanY*, double * *carY*, const int *inccarY*)

Add to manually specified indexed double precision Y the absolute sum of complex double precision vector X.

Add to Y the indexed sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.26 double idxdBLAS_dmzssq (const int *fold*, const int *N*, const void * *X*, const int *incX*, const double *scaleY*, double * *manY*, const int *incmanY*, double * *carY*, const int *inccarY*)

Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd_dscale\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.27 float idxdBLAS_samax (const int *N*, const float * *X*, const int *incX*)

Find maximum absolute value in vector of single precision.

Returns the absolute value of the element of maximum absolute value in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute maximum value of X

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.28 float idxdBLAS_samaxm (const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*)

Find maximum absolute value pairwise product between vectors of single precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

absolute maximum value multiple of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.29 void idxdBLAS_sicasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_indexed * *Y*)

Add to indexed single precision Y the absolute sum of complex single precision vector X.

Add to Y the indexed sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.30 float idxdBLAS_sicssq (const int *fold*, const int *N*, const void * *X*, const int *incX*, const float *scaleY*, float_indexed * *Y*)

Add to scaled indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>Y</i>	indexed scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.31 void idxdBLAS_sisasum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_indexed * *Y*)

Add to indexed single precision Y the absolute sum of single precision vector X.

Add to Y the indexed sum of absolute values of elements in X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.32 void idxdBLAS_sisdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*, float_indexed * *Z*)

Add to indexed single precision Z the dot product of single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length

<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	indexed scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.33 void idxdBLAS_sisgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float * *A*, const int *lda*, const float * *B*, const int *ldb*, float_indexed * *C*, const int *ldc*)

Add to indexed single precision matrix C the matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α is a scalar, A and B are matrices with $\text{op}(A)$ an M by K matrix and $\text{op}(B)$ a K by N matrix, and C is an indexed M by N matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	indexed single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.34 void idxdBLAS_sisgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, float_indexed * *Y*, const int *incY*)

Add to indexed single precision vector *Y* the matrix-vector product of single precision matrix *A* and single precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where α and β are scalars, x is a vector, y is an indexed vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar α
<i>A</i>	single precision matrix of dimension (M , lda) in row-major or (lda , N) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	<i>X</i> vector stride (use every $incX$ 'th element)
<i>beta</i>	scalar β
<i>Y</i>	indexed single precision vector <i>Y</i> of at least size M if not transposed or size N otherwise
<i>incY</i>	<i>Y</i> vector stride (use every $incY$ 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.35 float idxdBLAS_sissq (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float *scaleY*, float_indexed * *Y*)

Add to scaled indexed single precision *Y* the scaled sum of squares of elements of single precision vector *X*.

Add to *Y* the scaled indexed sum of the squares of each element of *X*. The scaling of each square is performed using [idxd_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.36 void idxdBLAS_sissum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_indexed * *Y*)

Add to indexed single precision Y the sum of single precision vector X.

Add to Y the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.37 void idxdBLAS_smcasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float * *manY*, const int *incmanY*, float * *cary*, const int *inccary*)

Add to manually specified indexed single precision Y the absolute sum of complex single precision vector X.

Add to Y the indexed sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)

<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.38 `float idxdBLAS_smcssq (const int fold, const int N, const void * X, const int incX, const float scaleY, float * manY, const int incmanY, float * carY, const int inccarY)`

Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.39 `void idxdBLAS_smsasum (const int fold, const int N, const float * X, const int incX, float * manY, const int incmanY, float * carY, const int inccarY)`

Add to manually specified indexed single precision Y the absolute sum of double precision vector X.

Add to Y to the indexed sum of absolute values of elements in X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.40 void idxdBLAS_smsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*, float * *manZ*, const int *incmanZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified indexed single precision Z the dot product of single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every incY'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every incpriZ'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every inccarZ'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.41 float idxdBLAS_smsssq (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float *scaleY*, float * *manY*, const int *incmanY*, float * *carY*, const int *inccarY*)

Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of single precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.42 void idxdBLAS_ssmsum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float * *manY*, const int *incmanY*, float * *carY*, const int *inccarY*)

Add to manually specified indexed single precision Y the sum of single precision vector X.

Add to Y the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.43 void idxdBLAS_zamax_sub (const int *N*, const void * *X*, const int *incX*, void * *amax*)

Find maximum magnitude in vector of complex double precision.

Returns the magnitude of the element of maximum magnitude in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>amax</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.44 void idxdBLAS_zamaxm_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *amaxm*)

Find maximum magnitude pairwise product between vectors of complex double precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>amaxm</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.45 void idxdBLAS_zizdotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double_complex_indexed * *Z*)

Add to indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>scalar</i>	return <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.46 void idxdBLAS_zizdotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double_complex_indexed * *Z*)

Add to indexed complex double precision *Z* the unconjugated dot product of complex double precision vectors *X* and *Y*.

Add to *Z* the indexed sum of the pairwise products of *X* and *Y*.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>indexed</i>	scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.47 void idxdBLAS_zizgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, double_complex_indexed * *C*, const int *ldc*)

Add to indexed complex double precision matrix *C* the matrix-matrix product of complex double precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

α is a scalar, *A* and *B* are matrices with $\text{op}(A)$ an *M* by *K* matrix and $\text{op}(B)$ a *K* by *N* matrix, and *C* is an indexed *M* by *N* matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	indexed complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

```
2.2.2.48 void idxdBLAS_zizgemv ( const int fold, const char Order, const char TransA, const int M, const int N, const void *
    alpha, const void * A, const int lda, const void * X, const int incX, double_complex_indexed * Y, const int incY
    )
```

Add to indexed complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A * T * x + \beta * y$ or $y := \alpha * A * H * x + \beta * y$,

where alpha and beta are scalars, x is a vector, y is an indexed vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A

<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	indexed complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.49 void idxdBLAS_zizsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double_complex_indexed * *Y*)

Add to indexed complex double precision Y the sum of complex double precision vector X.

Add to Y the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>indexed</i>	scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.50 void idxdBLAS_zmzdote (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double * *manZ*, const int *incmanZ*, double * *carZ*, const int *inccarZ*)

Add to manually specified indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.51 void idxdBLAS_zmzdotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double * *manZ*, const int *incmanZ*, double * *carZ*, const int *inccarZ*)

Add to manually specified indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.

Add to Z to the indexed sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.52 void idxdBLAS_zmzsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double * *manY*, const int *incmanY*, double * *carY*, const int *inccarY*)

Add to manually specified indexed complex double precision Y the sum of complex double precision vector X.

Add to Y the indexed sum of X.

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.3 include/reproBLAS.h File Reference

[reproBLAS.h](#) defines reproducible BLAS Methods.

```
#include <complex.h>
```

Functions

- double [reproBLAS_rdsun](#) (const int fold, const int N, const double *X, const int incX)
Compute the reproducible sum of double precision vector X.
- double [reproBLAS_rdasun](#) (const int fold, const int N, const double *X, const int incX)
Compute the reproducible absolute sum of double precision vector X.
- double [reproBLAS_rdnrm2](#) (const int fold, const int N, const double *X, const int incX)
Compute the reproducible Euclidian norm of double precision vector X.
- double [reproBLAS_rddot](#) (const int fold, const int N, const double *X, const int incX, const double *Y, const int incY)
Compute the reproducible dot product of double precision vectors X and Y.
- float [reproBLAS_rsdot](#) (const int fold, const int N, const float *X, const int incX, const float *Y, const int incY)
Compute the reproducible dot product of single precision vectors X and Y.
- float [reproBLAS_rsasun](#) (const int fold, const int N, const float *X, const int incX)
Compute the reproducible absolute sum of single precision vector X.
- float [reproBLAS_rssun](#) (const int fold, const int N, const float *X, const int incX)
Compute the reproducible sum of single precision vector X.
- float [reproBLAS_rsnrm2](#) (const int fold, const int N, const float *X, const int incX)
Compute the reproducible Euclidian norm of single precision vector X.
- void [reproBLAS_rzsum_sub](#) (const int fold, const int N, const void *X, int incX, void *sum)
Compute the reproducible sum of complex double precision vector X.
- double [reproBLAS_rdzasun](#) (const int fold, const int N, const void *X, const int incX)
Compute the reproducible absolute sum of complex double precision vector X.
- double [reproBLAS_rdznm2](#) (const int fold, const int N, const void *X, int incX)
Compute the reproducible Euclidian norm of complex double precision vector X.
- void [reproBLAS_rzdotc_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)

Compute the reproducible conjugated dot product of complex double precision vectors X and Y.

- void [reproBLAS_rzdotu_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)

Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.

- void [reproBLAS_rcsum_sub](#) (const int fold, const int N, const void *X, const int incX, void *sum)

Compute the reproducible sum of complex single precision vector X.

- float [reproBLAS_rscasum](#) (const int fold, const int N, const void *X, const int incX)

Compute the reproducible absolute sum of complex single precision vector X.

- float [reproBLAS_rscnrm2](#) (const int fold, const int N, const void *X, const int incX)

Compute the reproducible Euclidian norm of complex single precision vector X.

- void [reproBLAS_rcdotc_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)

Compute the reproducible conjugated dot product of complex single precision vectors X and Y.

- void [reproBLAS_rcdotu_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)

Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.

- void [reproBLAS_rdgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.

- void [reproBLAS_rdgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

- void [reproBLAS_rsgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)

Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.

- void [reproBLAS_rsgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)

Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.

- void [reproBLAS_rzgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.

- void [reproBLAS_rzgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

- void [reproBLAS_rcgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

- void [reproBLAS_rcgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

- double [reproBLAS_dsum](#) (const int N, const double *X, const int incX)

- Compute the reproducible sum of double precision vector X.*
- double [reproBLAS_dasum](#) (const int N, const double *X, const int incX)
- Compute the reproducible absolute sum of double precision vector X.*
- double [reproBLAS_dnrm2](#) (const int N, const double *X, const int incX)
- Compute the reproducible Euclidian norm of double precision vector X.*
- double [reproBLAS_ddot](#) (const int N, const double *X, const int incX, const double *Y, const int incY)
- Compute the reproducible dot product of double precision vectors X and Y.*
- float [reproBLAS_sdot](#) (const int N, const float *X, const int incX, const float *Y, const int incY)
- Compute the reproducible dot product of single precision vectors X and Y.*
- float [reproBLAS_sasum](#) (const int N, const float *X, const int incX)
- Compute the reproducible absolute sum of single precision vector X.*
- float [reproBLAS_ssum](#) (const int N, const float *X, const int incX)
- Compute the reproducible sum of single precision vector X.*
- float [reproBLAS_snrm2](#) (const int N, const float *X, const int incX)
- Compute the reproducible Euclidian norm of single precision vector X.*
- void [reproBLAS_zsum_sub](#) (const int N, const void *X, int incX, void *sum)
- Compute the reproducible sum of complex double precision vector X.*
- double [reproBLAS_dzasum](#) (const int N, const void *X, const int incX)
- Compute the reproducible absolute sum of complex double precision vector X.*
- double [reproBLAS_dznrm2](#) (const int N, const void *X, int incX)
- Compute the reproducible Euclidian norm of complex double precision vector X.*
- void [reproBLAS_zdotc_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)
- Compute the reproducible conjugated dot product of complex double precision vectors X and Y.*
- void [reproBLAS_zdotu_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)
- Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.*
- void [reproBLAS_csum_sub](#) (const int N, const void *X, const int incX, void *sum)
- Compute the reproducible sum of complex single precision vector X.*
- float [reproBLAS_scasum](#) (const int N, const void *X, const int incX)
- Compute the reproducible absolute sum of complex single precision vector X.*
- float [reproBLAS_scnrm2](#) (const int N, const void *X, const int incX)
- Compute the reproducible Euclidian norm of complex single precision vector X.*
- void [reproBLAS_cdotc_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)
- Compute the reproducible conjugated dot product of complex single precision vectors X and Y.*
- void [reproBLAS_cdotu_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)
- Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.*
- void [reproBLAS_dgemv](#) (const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)
- Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.*
- void [reproBLAS_dgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)
- Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.*
- void [reproBLAS_sgemv](#) (const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)
- Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.*
- void [reproBLAS_sgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)

- Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.*
- void [reproBLAS_zgemv](#) (const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)
- Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.*
- void [reproBLAS_zgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)
- Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.*
- void [reproBLAS_cgemv](#) (const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)
- Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.*
- void [reproBLAS_cgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)
- Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.*

2.3.1 Detailed Description

[reproBLAS.h](#) defines reproducible BLAS Methods.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by `sub`, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Indexed types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many bins are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in `config.h`. If you are unsure of what value to use for , we recommend 3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use.

In reproBLAS, two copies of the BLAS are provided. The functions that share the same name as their BLAS counterparts perform reproducible versions of their corresponding operations using the default fold value specified

in config.h. The functions that are prefixed by the character 'r' allow the user to specify their own fold for the underlying indexed types.

2.3.2 Function Documentation

2.3.2.1 void reproBLAS_cdotc_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotc*)

Compute the reproducible conjugated dot product of complex single precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and conjugated *Y*.

The reproducible dot product is computed with indexed types of default fold using [idxdBLAS_cicdotc\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.2 void reproBLAS_cdotu_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotu*)

Compute the reproducible unconjugated dot product of complex single precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with indexed types of default fold using [idxdBLAS_cicdotu\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.3 void reproBLAS_cgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$ or $\text{op}(X) = X^{**H}$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types of default fold with [idxdBLAS_cicgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.4 void reproBLAS_cgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^{**T} * x + \beta * y$ or $y := \alpha * A^{**H} * x + \beta * y$,

where α and β are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types of default fold with [idxdBLAS_cicgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (M, <i>lda</i>) in row-major or (<i>lda</i> , N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.5 `void reproBLAS_csum_sub (const int N, const void * X, const int incX, void * sum)`

Compute the reproducible sum of complex single precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types of default fold using [idxdBLAS_cicsum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.6 `double reproBLAS_dasum (const int N, const double * X, const int incX)`

Compute the reproducible absolute sum of double precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with indexed types of default fold using [idxdBLAS_didasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.7 double reproBLAS_ddot (const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*)

Compute the reproducible dot product of double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with indexed types of default fold using [idxdBLAS_diddot\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.8 void reproBLAS_dgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double * *A*, const int *lda*, const double * *B*, const int *ldb*, const double *beta*, double * *C*, const int *ldc*)

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

alpha and beta are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types of default fold with [idxdBLAS_didgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.9 void reproBLAS_dgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const double *alpha*, const double * *A*, const int *lda*, const double * *X*, const int *incX*, const double *beta*, double * *Y*, const int *incY*)

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types of default fold with [idxdBLAS_didgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major

<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>beta</i>	scalar beta
<i>Y</i>	double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.10 double reproBLAS_dnorm2 (const int *N*, const double * *X*, const int *incX*)

Compute the reproducible Euclidian norm of double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types of default fold using [idxdBLAS_didssq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.11 double reproBLAS_dsum (const int *N*, const double * *X*, const int *incX*)

Compute the reproducible sum of double precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types of default fold using [idxdBLAS_didsum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
-------------	---

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.12 double reproBLAS_dzasum (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex double precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with indexed types of default fold using [idxdBLAS_dzasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.13 double reproBLAS_dznrm2 (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types of default fold using [idxdBLAS_dizssq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.14 `void reproBLAS_rcdotc_sub (const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, void * dotc)`

Compute the reproducible conjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with indexed types using [idxdBLAS_cicdotc\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.15 `void reproBLAS_rcdotu_sub (const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, void * dotu)`

Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with indexed types using [idxdBLAS_cicdotu\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.16 void reproBLAS_rcgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types with [idxdBLAS_cicgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.17 void reproBLAS_rcgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^{**}T * x + \beta * y$ or $y := \alpha * A^{**}H * x + \beta * y$,

where α and β are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types with [idxdBLAS_cicgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (M, <i>lda</i>) in row-major or (<i>lda</i> , N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.18 `void reproBLAS_rcsum_sub (const int fold, const int N, const void * X, const int incX, void * sum)`

Compute the reproducible sum of complex single precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types using [idxdBLAS_cicsum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.19 `double reproBLAS_rdasum (const int fold, const int N, const double * X, const int incX)`

Compute the reproducible absolute sum of double precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with indexed types using [idxdBLAS_didasum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.20 `double reproBLAS_rddot (const int fold, const int N, const double * X, const int incX, const double * Y, const int incY)`

Compute the reproducible dot product of double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with indexed types using [idxdBLAS_diddot\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.21 `void reproBLAS_rdgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double * A, const int lda, const double * B, const int ldb, const double beta, double * C, const int ldc)`

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C,$

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types with [idxdBLAS_didgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C .
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C .
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	double precision matrix of dimension (ma , lda) in row-major or (lda , na) in column-major. (ma , na) is (M , K) if A is not transposed and (K , M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	double precision matrix of dimension (mb , ldb) in row-major or (ldb , nb) in column-major. (mb , nb) is (K , N) if B is not transposed and (N , K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	double precision matrix of dimension (M , ldc) in row-major or (ldc , N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.22 `void reproBLAS_rdgemv (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double * A, const int lda, const double * X, const int incX, const double beta, double * Y, const int incY)`

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X .

Performs one of the matrix-vector operations

$y := \alpha A x + \beta y$ or $y := \alpha A^{**T} x + \beta y$,

where α and β are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types with [idxdBLAS_didgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.23 double reproBLAS_rdnrm2 (const int *fold*, const int *N*, const double * *X*, const int *incX*)

Compute the reproducible Euclidian norm of double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types using [idxdBLAS_didssq\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.24 double reproBLAS_rdsun (const int *fold*, const int *N*, const double * *X*, const int *incX*)

Compute the reproducible sum of double precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types using [idxdBLAS_didsun\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.25 double reproBLAS_rdzasum (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex double precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with indexed types using [idxdBLAS_dizasum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.26 double reproBLAS_rdnrm2 (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types using [idxdBLAS_dizssq\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.27 float reproBLAS_rsasum (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible absolute sum of single precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with indexed types using [idxdBLAS_sisasum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.28 float reproBLAS_rscasum (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex single precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with indexed types using [idxdBLAS_sicasum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.29 float reproBLAS_rscnrm2 (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types using [idxdBLAS_sicssq\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.30 float reproBLAS_rsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*)

Compute the reproducible dot product of single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with indexed types using [idxdBLAS_sisdot\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every incY'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.31 `void reproBLAS_rsgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float * A, const int lda, const float * B, const int ldb, const float beta, float * C, const int ldc)`

Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where op(X) is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types with [idxdBLAS_sisgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.

<i>lda</i>	the first dimension of A as declared in the calling program. <i>lda</i> must be at least <i>na</i> in row major or <i>ma</i> in column major.
<i>B</i>	single precision matrix of dimension (<i>mb</i> , <i>ldb</i>) in row-major or (<i>ldb</i> , <i>nb</i>) in column-major. (<i>mb</i> , <i>nb</i>) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. <i>ldb</i> must be at least <i>nb</i> in row major or <i>mb</i> in column major.
<i>C</i>	single precision matrix of dimension (M, <i>ldc</i>) in row-major or (<i>ldc</i> , N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. <i>ldc</i> must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.32 `void reproBLAS_rsgemv (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float * A, const int lda, const float * X, const int incX, const float beta, float * Y, const int incY)`

Add to single precision vector *Y* the reproducible matrix-vector product of single precision matrix *A* and single precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where α and β are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types with [idxdBLAS_sisgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (M, <i>lda</i>) in row-major or (<i>lda</i> , N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>beta</i>	scalar beta
<i>Y</i>	single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.33 float reproBLAS_rsnrm2 (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible Euclidian norm of single precision vector *X*.

Return the square root of the sum of the squared elements of *X*.

The reproducible Euclidian norm is computed with scaled indexed types using [idxdBLAS_sisssq\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of *X*

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.34 float reproBLAS_rssum (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible sum of single precision vector *X*.

Return the sum of *X*.

The reproducible sum is computed with indexed types using [idxdBLAS_sissum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

Returns

sum of *X*

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.35 void reproBLAS_rzdotc_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotc*)

Compute the reproducible conjugated dot product of complex double precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and conjugated *Y*.

The reproducible dot product is computed with indexed types using [idxdBLAS_zizdotc\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.36 void reproBLAS_rzdotu_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotu*)

Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with indexed types using [idxdBLAS_zizdotu\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.37 void reproBLAS_rzgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

alpha and beta are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types with [idxdBLAS_zizgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.38 void reproBLAS_rzgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha A x + \beta y$ or $y := \alpha A^T x + \beta y$ or $y := \alpha A^H x + \beta y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types with [idxdBLAS_zizgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A

<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.39 void reproBLAS_rzsum_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, void * *sum*)

Compute the reproducible sum of complex double precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types using [idxdBLAS_zizsum\(\)](#)

Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.40 float reproBLAS_sasum (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible absolute sum of single precision vector X.

Return the sum of absolute values of elements in X.

The reproducible absolute sum is computed with indexed types of default fold using [idxdBLAS_sisasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
-------------	---

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.41 float reproBLAS_scasum (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex single precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with indexed types of default fold using [idxdBLAS_sicasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.42 float reproBLAS_scnrm2 (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types of default fold using [idxdBLAS_sicssq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.43 float reproBLAS_sdot (const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*)

Compute the reproducible dot product of single precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with indexed types of default fold using [idxdBLAS_sisdot\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of *X* and *Y*

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.44 void reproBLAS_sgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float * *A*, const int *lda*, const float * *B*, const int *ldb*, const float *beta*, float * *C*, const int *ldc*)

Add to single precision matrix *C* the reproducible matrix-matrix product of single precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α and β are scalars, *A* and *B* and *C* are matrices with $\text{op}(A)$ an *M* by *K* matrix, $\text{op}(B)$ a *K* by *N* matrix, and *C* is an *M* by *N* matrix.

The matrix-matrix product is computed using indexed types of default fold with [idxdBLAS_sisgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.45 void reproBLAS_sgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, const float *beta*, float * *Y*, const int *incY*)

Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types of default fold with [idxdBLAS_sisgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	single precision vector of at least size N if not transposed or size M otherwise

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>beta</i>	scalar beta
<i>Y</i>	single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.46 float reproBLAS_snrm2 (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible Euclidian norm of single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled indexed types of default fold using [idxdBLAS_sissq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.47 float reproBLAS_ssum (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible sum of single precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types of default fold using [idxdBLAS_sissum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.48 `void reproBLAS_zdotc_sub (const int N, const void * X, const int incX, const void * Y, const int incY, void * dotc)`

Compute the reproducible conjugated dot product of complex double precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and conjugated *Y*.

The reproducible dot product is computed with indexed types of default fold using [idxdBLAS_zizdotc\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.49 `void reproBLAS_zdotu_sub (const int N, const void * X, const int incX, const void * Y, const int incY, void * dotu)`

Compute the reproducible unconjugated dot product of complex double precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with indexed types of default fold using [idxdBLAS_zizdotu\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.3.2.50 void reproBLAS_zgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where op(X) is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using indexed types of default fold with [idxdBLAS_zizgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.51 void reproBLAS_zgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^{**}T * x + \beta * y$ or $y := \alpha * A^{**}H * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using indexed types of default fold with [idxdBLAS_zizgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.3.2.52 `void reproBLAS_zsum_sub (const int N, const void * X, const int incX, void * sum)`

Compute the reproducible sum of complex double precision vector X.

Return the sum of X.

The reproducible sum is computed with indexed types of default fold using [idxdBLAS_zizsum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

Index

DIWIDTH

[idxd.h, 11](#)

double_complex_indexed

[idxd.h, 14](#)

double_indexed

[idxd.h, 14](#)

float_complex_indexed

[idxd.h, 15](#)

float_indexed

[idxd.h, 15](#)

idxd.h

[DIWIDTH, 11](#)

[double_complex_indexed, 14](#)

[double_indexed, 14](#)

[float_complex_indexed, 15](#)

[float_indexed, 15](#)

[idxd_DICAPACITY, 11](#)

[idxd_DIENDURANCE, 11](#)

[idxd_DIMAXFOLD, 11](#)

[idxd_DIMAXINDEX, 12](#)

[idxd_DMCOMPRESSION, 12](#)

[idxd_DMEXPANSION, 12](#)

[idxd_SICAPACITY, 12](#)

[idxd_SIENDURANCE, 13](#)

[idxd_SIMAXFOLD, 13](#)

[idxd_SIMAXINDEX, 13](#)

[idxd_SMCOMPRESSION, 13](#)

[idxd_SMEXPANSION, 14](#)

[idxd_ccconv_sub, 15](#)

[idxd_ccmconv_sub, 15](#)

[idxd_cialloc, 17](#)

[idxd_cicadd, 17](#)

[idxd_cicconv, 17](#)

[idxd_cicdeposit, 19](#)

[idxd_ciciadd, 19](#)

[idxd_ciciaddv, 20](#)

[idxd_ciciset, 20](#)

[idxd_cicupdate, 20](#)

[idxd_cinegate, 21](#)

[idxd_cinum, 21](#)

[idxd_ciprint, 21](#)

[idxd_cirenorm, 23](#)

[idxd_cisetzero, 23](#)

[idxd_cisiset, 23](#)

[idxd_cisize, 24](#)

[idxd_cisupdate, 24](#)

[idxd_cmadd, 24](#)

[idxd_cmconv, 26](#)

[idxd_cmcddeposit, 26](#)

[idxd_cmcmadd, 27](#)

[idxd_cmcmset, 27](#)

[idxd_cmupdate, 28](#)

[idxd_cmdenorm, 28](#)

[idxd_cmnegate, 29](#)

[idxd_cmprint, 29](#)

[idxd_cmrenorm, 29](#)

[idxd_cmsetzero, 30](#)

[idxd_cmsmset, 30](#)

[idxd_cmsrescale, 31](#)

[idxd_cmsupdate, 31](#)

[idxd_ddconv, 32](#)

[idxd_ddmconv, 32](#)

[idxd_dialloc, 33](#)

[idxd_dibound, 33](#)

[idxd_didadd, 33](#)

[idxd_didconv, 34](#)

[idxd_diddeposit, 34](#)

[idxd_didiadd, 35](#)

[idxd_didiaddsq, 35](#)

[idxd_didiaddv, 35](#)

[idxd_didiset, 36](#)

[idxd_didupdate, 36](#)

[idxd_dindex, 37](#)

[idxd_dinegate, 37](#)

[idxd_dinum, 37](#)

[idxd_diprint, 38](#)

[idxd_direnorm, 38](#)

[idxd_disetzero, 38](#)

[idxd_disize, 39](#)

[idxd_dmbins, 39](#)

[idxd_dmdadd, 39](#)

[idxd_dmdconv, 41](#)

[idxd_dmddeposit, 41](#)

[idxd_dmdenorm, 42](#)

[idxd_dmdmadd, 42](#)

[idxd_dmdmaddsq, 43](#)

[idxd_dmdmset, 43](#)

[idxd_dmdrescale, 44](#)

[idxd_dmdupdate, 44](#)

[idxd_dmindex, 45](#)

[idxd_dmindex0, 45](#)

[idxd_dmnegate, 45](#)

[idxd_dmprint, 46](#)

[idxd_dmrenorm, 46](#)

[idxd_dmsetzero, 47](#)

[idxd_dscale, 47](#)

[idxd_sialloc, 48](#)

idxd_sibound, 48
 idxd_sindex, 48
 idxd_sinegate, 49
 idxd_sinum, 49
 idxd_siprint, 50
 idxd_sirenorm, 50
 idxd_sisadd, 50
 idxd_sisconv, 51
 idxd_sisdeposit, 51
 idxd_sisetzzero, 52
 idxd_sisiadd, 52
 idxd_sisiaddsq, 52
 idxd_sisiadv, 53
 idxd_sisiset, 53
 idxd_sisize, 54
 idxd_sisupdate, 54
 idxd_smbins, 54
 idxd_smdenorm, 55
 idxd_smindex, 55
 idxd_smindex0, 55
 idxd_smnegate, 56
 idxd_smprint, 56
 idxd_smrenorm, 57
 idxd_smsadd, 57
 idxd_smsconv, 57
 idxd_smsdeposit, 59
 idxd_smsetzero, 59
 idxd_smsmadd, 60
 idxd_smsmaddsq, 60
 idxd_smsmset, 61
 idxd_smsrescale, 61
 idxd_smsupdate, 62
 idxd_ssacle, 62
 idxd_ssiconv, 63
 idxd_ssmconv, 63
 idxd_ufp, 64
 idxd_ufpf, 64
 idxd_zialloc, 64
 idxd_zidiset, 65
 idxd_zidupdate, 65
 idxd_zinegate, 65
 idxd_zinum, 67
 idxd_ziprint, 67
 idxd_zirenorm, 67
 idxd_zisetzzero, 68
 idxd_zisize, 68
 idxd_zizadd, 68
 idxd_zizconv, 69
 idxd_zizdeposit, 69
 idxd_ziziadd, 70
 idxd_ziziadv, 70
 idxd_ziziset, 70
 idxd_zizupdate, 72
 idxd_zmdenorm, 72
 idxd_zmdmset, 72
 idxd_zmdrescale, 73
 idxd_zmdupdate, 73
 idxd_zmnegate, 75
 idxd_zmprint, 75
 idxd_zmrenorm, 76
 idxd_zmsetzero, 76
 idxd_zmzadd, 76
 idxd_zmzconv, 77
 idxd_zmzdeposit, 77
 idxd_zmzmadd, 78
 idxd_zmzmset, 78
 idxd_zmzupdate, 79
 idxd_zziconv_sub, 79
 idxd_zzmconv_sub, 80
 SIWIDTH, 14
 idxd_DICAPACITY
 idxd.h, 11
 idxd_DIENDURANCE
 idxd.h, 11
 idxd_DIMAXFOLD
 idxd.h, 11
 idxd_DIMAXINDEX
 idxd.h, 12
 idxd_DMCOMPRESSION
 idxd.h, 12
 idxd_DMEXPANSION
 idxd.h, 12
 idxd_SICAPACITY
 idxd.h, 12
 idxd_SIENDURANCE
 idxd.h, 13
 idxd_SIMAXFOLD
 idxd.h, 13
 idxd_SIMAXINDEX
 idxd.h, 13
 idxd_SMCOMPRESSION
 idxd.h, 13
 idxd_SMEXPANSION
 idxd.h, 14
 idxd_cciconv_sub
 idxd.h, 15
 idxd_ccmconv_sub
 idxd.h, 15
 idxd_cialloc
 idxd.h, 17
 idxd_cicadd
 idxd.h, 17
 idxd_cicconv
 idxd.h, 17
 idxd_cicdeposit
 idxd.h, 19
 idxd_ciciadd
 idxd.h, 19
 idxd_ciciadv
 idxd.h, 20
 idxd_ciciset
 idxd.h, 20
 idxd_cicupdate
 idxd.h, 20
 idxd_cinegate
 idxd.h, 21

idxd_cinum
 idxd.h, [21](#)

idxd_ciprint
 idxd.h, [21](#)

idxd_cirenorm
 idxd.h, [23](#)

idxd_cisetzero
 idxd.h, [23](#)

idxd_cisiset
 idxd.h, [23](#)

idxd_cisize
 idxd.h, [24](#)

idxd_cisupdate
 idxd.h, [24](#)

idxd_cmcadd
 idxd.h, [24](#)

idxd_cmconv
 idxd.h, [26](#)

idxd_cmdeposit
 idxd.h, [26](#)

idxd_cmcmadd
 idxd.h, [27](#)

idxd_cmcmset
 idxd.h, [27](#)

idxd_cmcupdate
 idxd.h, [28](#)

idxd_cmddenorm
 idxd.h, [28](#)

idxd_cmnegate
 idxd.h, [29](#)

idxd_cmprint
 idxd.h, [29](#)

idxd_cmrenorm
 idxd.h, [29](#)

idxd_cmsetzero
 idxd.h, [30](#)

idxd_cmsmset
 idxd.h, [30](#)

idxd_cmsrescale
 idxd.h, [31](#)

idxd_cmsupdate
 idxd.h, [31](#)

idxd_ddiconv
 idxd.h, [32](#)

idxd_ddmconv
 idxd.h, [32](#)

idxd_dialloc
 idxd.h, [33](#)

idxd_dibound
 idxd.h, [33](#)

idxd_didadd
 idxd.h, [33](#)

idxd_didconv
 idxd.h, [34](#)

idxd_diddeposit
 idxd.h, [34](#)

idxd_didiadd
 idxd.h, [35](#)

idxd_didiaddsq
 idxd.h, [35](#)

idxd_didiadv
 idxd.h, [35](#)

idxd_didiset
 idxd.h, [36](#)

idxd_didupdate
 idxd.h, [36](#)

idxd_dindex
 idxd.h, [37](#)

idxd_dinegate
 idxd.h, [37](#)

idxd_dinum
 idxd.h, [37](#)

idxd_diprint
 idxd.h, [38](#)

idxd_direnorm
 idxd.h, [38](#)

idxd_disetzero
 idxd.h, [38](#)

idxd_disize
 idxd.h, [39](#)

idxd_dmbins
 idxd.h, [39](#)

idxd_dmdadd
 idxd.h, [39](#)

idxd_dmdconv
 idxd.h, [41](#)

idxd_dmddeposit
 idxd.h, [41](#)

idxd_dmdenorm
 idxd.h, [42](#)

idxd_dmdmadd
 idxd.h, [42](#)

idxd_dmdmaddsq
 idxd.h, [43](#)

idxd_dmdmset
 idxd.h, [43](#)

idxd_dmdrescale
 idxd.h, [44](#)

idxd_dmdupdate
 idxd.h, [44](#)

idxd_dmindex
 idxd.h, [45](#)

idxd_dmindex0
 idxd.h, [45](#)

idxd_dmnegate
 idxd.h, [45](#)

idxd_dmprint
 idxd.h, [46](#)

idxd_dmrenorm
 idxd.h, [46](#)

idxd_dmsetzero
 idxd.h, [47](#)

idxd_dscale
 idxd.h, [47](#)

idxd_sialloc
 idxd.h, [48](#)

idxd_sibound	idxd_smsmset
idxd.h, 48	idxd.h, 61
idxd_sindex	idxd_smsrescale
idxd.h, 48	idxd.h, 61
idxd_sinegate	idxd_smsupdate
idxd.h, 49	idxd.h, 62
idxd_sinum	idxd_ssacle
idxd.h, 49	idxd.h, 62
idxd_siprint	idxd_ssiconv
idxd.h, 50	idxd.h, 63
idxd_sirenorm	idxd_ssmconv
idxd.h, 50	idxd.h, 63
idxd_sisadd	idxd_ufp
idxd.h, 50	idxd.h, 64
idxd_sisconv	idxd_ufpf
idxd.h, 51	idxd.h, 64
idxd_sisdeposit	idxd_zialloc
idxd.h, 51	idxd.h, 64
idxd_sisetzero	idxd_zidiset
idxd.h, 52	idxd.h, 65
idxd_sisiadd	idxd_zidupdate
idxd.h, 52	idxd.h, 65
idxd_sisiaddsq	idxd_zinegate
idxd.h, 52	idxd.h, 65
idxd_sisiaddv	idxd_zinum
idxd.h, 53	idxd.h, 67
idxd_sisiset	idxd_ziprint
idxd.h, 53	idxd.h, 67
idxd_sisize	idxd_zirenorm
idxd.h, 54	idxd.h, 67
idxd_sisupdate	idxd_zisetzero
idxd.h, 54	idxd.h, 68
idxd_smbins	idxd_zisize
idxd.h, 54	idxd.h, 68
idxd_smdenorm	idxd_zizadd
idxd.h, 55	idxd.h, 68
idxd_smindex	idxd_zizconv
idxd.h, 55	idxd.h, 69
idxd_smindex0	idxd_zizdeposit
idxd.h, 55	idxd.h, 69
idxd_smnegate	idxd_ziziadd
idxd.h, 56	idxd.h, 70
idxd_smprint	idxd_ziziaddv
idxd.h, 56	idxd.h, 70
idxd_smrenorm	idxd_ziziset
idxd.h, 57	idxd.h, 70
idxd_smsadd	idxd_zizupdate
idxd.h, 57	idxd.h, 72
idxd_smsconv	idxd_zmdenorm
idxd.h, 57	idxd.h, 72
idxd_smsdeposit	idxd_zmdmset
idxd.h, 59	idxd.h, 72
idxd_smsetzero	idxd_zmdrescale
idxd.h, 59	idxd.h, 73
idxd_smsmadd	idxd_zmdupdate
idxd.h, 60	idxd.h, 73
idxd_smsmaddsq	idxd_zmnegate
idxd.h, 60	idxd.h, 75

- idxd_zmprint
 - idxd.h, [75](#)
- idxd_zmrenorm
 - idxd.h, [76](#)
- idxd_zmsetzero
 - idxd.h, [76](#)
- idxd_zmzadd
 - idxd.h, [76](#)
- idxd_zmzconv
 - idxd.h, [77](#)
- idxd_zmzdeposit
 - idxd.h, [77](#)
- idxd_zmzmadd
 - idxd.h, [78](#)
- idxd_zmzmset
 - idxd.h, [78](#)
- idxd_zmzupdate
 - idxd.h, [79](#)
- idxd_zziconv_sub
 - idxd.h, [79](#)
- idxd_zzmconv_sub
 - idxd.h, [80](#)
- idxdBLAS.h
 - idxdBLAS_camax_sub, [84](#)
 - idxdBLAS_camaxm_sub, [85](#)
 - idxdBLAS_cicdotc, [85](#)
 - idxdBLAS_cicdotu, [86](#)
 - idxdBLAS_cicgemm, [86](#)
 - idxdBLAS_cicgemv, [87](#)
 - idxdBLAS_cicsum, [88](#)
 - idxdBLAS_cmcdotc, [88](#)
 - idxdBLAS_cmcdotu, [89](#)
 - idxdBLAS_cmcsu, [89](#)
 - idxdBLAS_damax, [90](#)
 - idxdBLAS_damaxm, [90](#)
 - idxdBLAS_didasum, [91](#)
 - idxdBLAS_diddot, [91](#)
 - idxdBLAS_didgemm, [92](#)
 - idxdBLAS_didgemv, [92](#)
 - idxdBLAS_didssq, [93](#)
 - idxdBLAS_didsum, [94](#)
 - idxdBLAS_dizasum, [94](#)
 - idxdBLAS_dizssq, [94](#)
 - idxdBLAS_dmdasum, [95](#)
 - idxdBLAS_dmddot, [95](#)
 - idxdBLAS_dmdssq, [96](#)
 - idxdBLAS_dmdsum, [96](#)
 - idxdBLAS_dmzasum, [97](#)
 - idxdBLAS_dmzssq, [97](#)
 - idxdBLAS_samax, [98](#)
 - idxdBLAS_samaxm, [98](#)
 - idxdBLAS_sicasum, [99](#)
 - idxdBLAS_sicssq, [99](#)
 - idxdBLAS_sisasum, [100](#)
 - idxdBLAS_sisdot, [100](#)
 - idxdBLAS_sisgemm, [101](#)
 - idxdBLAS_sisgemv, [102](#)
 - idxdBLAS_sissq, [102](#)
 - idxdBLAS_sissu, [103](#)
 - idxdBLAS_smcasum, [103](#)
 - idxdBLAS_smcssq, [104](#)
 - idxdBLAS_smsasum, [104](#)
 - idxdBLAS_smsdot, [105](#)
 - idxdBLAS_smssq, [105](#)
 - idxdBLAS_smssu, [106](#)
 - idxdBLAS_zamax_sub, [106](#)
 - idxdBLAS_zamaxm_sub, [107](#)
 - idxdBLAS_zizdotc, [107](#)
 - idxdBLAS_zizdotu, [108](#)
 - idxdBLAS_zizgemm, [108](#)
 - idxdBLAS_zizgemv, [109](#)
 - idxdBLAS_zizsum, [110](#)
 - idxdBLAS_zmzdotc, [110](#)
 - idxdBLAS_zmzdotu, [111](#)
 - idxdBLAS_zmzsum, [111](#)
- idxdBLAS_camax_sub
 - idxdBLAS.h, [84](#)
- idxdBLAS_camaxm_sub
 - idxdBLAS.h, [85](#)
- idxdBLAS_cicdotc
 - idxdBLAS.h, [85](#)
- idxdBLAS_cicdotu
 - idxdBLAS.h, [86](#)
- idxdBLAS_cicgemm
 - idxdBLAS.h, [86](#)
- idxdBLAS_cicgemv
 - idxdBLAS.h, [87](#)
- idxdBLAS_cicsum
 - idxdBLAS.h, [88](#)
- idxdBLAS_cmcdotc
 - idxdBLAS.h, [88](#)
- idxdBLAS_cmcdotu
 - idxdBLAS.h, [89](#)
- idxdBLAS_cmcsu
 - idxdBLAS.h, [89](#)
- idxdBLAS_damax
 - idxdBLAS.h, [90](#)
- idxdBLAS_damaxm
 - idxdBLAS.h, [90](#)
- idxdBLAS_didasum
 - idxdBLAS.h, [91](#)
- idxdBLAS_diddot
 - idxdBLAS.h, [91](#)
- idxdBLAS_didgemm
 - idxdBLAS.h, [92](#)
- idxdBLAS_didgemv
 - idxdBLAS.h, [92](#)
- idxdBLAS_didssq
 - idxdBLAS.h, [93](#)
- idxdBLAS_didsum
 - idxdBLAS.h, [94](#)
- idxdBLAS_dizasum
 - idxdBLAS.h, [94](#)
- idxdBLAS_dizssq
 - idxdBLAS.h, [94](#)
- idxdBLAS_dmdasum

- idxdBLAS.h, 95
- idxdBLAS_dmddot
 - idxdBLAS.h, 95
- idxdBLAS_dmdssq
 - idxdBLAS.h, 96
- idxdBLAS_dmdsum
 - idxdBLAS.h, 96
- idxdBLAS_dmzasum
 - idxdBLAS.h, 97
- idxdBLAS_dmzssq
 - idxdBLAS.h, 97
- idxdBLAS_samax
 - idxdBLAS.h, 98
- idxdBLAS_samaxm
 - idxdBLAS.h, 98
- idxdBLAS_sicasum
 - idxdBLAS.h, 99
- idxdBLAS_sicssq
 - idxdBLAS.h, 99
- idxdBLAS_sisasum
 - idxdBLAS.h, 100
- idxdBLAS_sisdot
 - idxdBLAS.h, 100
- idxdBLAS_sisgemm
 - idxdBLAS.h, 101
- idxdBLAS_sisgemv
 - idxdBLAS.h, 102
- idxdBLAS_sisssq
 - idxdBLAS.h, 102
- idxdBLAS_sissum
 - idxdBLAS.h, 103
- idxdBLAS_smcasum
 - idxdBLAS.h, 103
- idxdBLAS_smcssq
 - idxdBLAS.h, 104
- idxdBLAS_smsasum
 - idxdBLAS.h, 104
- idxdBLAS_smsdot
 - idxdBLAS.h, 105
- idxdBLAS_smsssq
 - idxdBLAS.h, 105
- idxdBLAS_smssum
 - idxdBLAS.h, 106
- idxdBLAS_zamax_sub
 - idxdBLAS.h, 106
- idxdBLAS_zamaxm_sub
 - idxdBLAS.h, 107
- idxdBLAS_zizdotc
 - idxdBLAS.h, 107
- idxdBLAS_zizdotu
 - idxdBLAS.h, 108
- idxdBLAS_zizgemm
 - idxdBLAS.h, 108
- idxdBLAS_zizgemv
 - idxdBLAS.h, 109
- idxdBLAS_zizsum
 - idxdBLAS.h, 110
- idxdBLAS_zmzdotc
 - idxdBLAS.h, 110
- idxdBLAS_zmzdotu
 - idxdBLAS.h, 111
- idxdBLAS_zmzsum
 - idxdBLAS.h, 111
- include/idxd.h, 3
- include/idxdBLAS.h, 80
- include/reproBLAS.h, 112
- reproBLAS.h
 - reproBLAS_cdotc_sub, 116
 - reproBLAS_cdotu_sub, 116
 - reproBLAS_cgemm, 116
 - reproBLAS_cgmv, 117
 - reproBLAS_csum_sub, 118
 - reproBLAS_dasum, 118
 - reproBLAS_ddot, 119
 - reproBLAS_dgemm, 119
 - reproBLAS_dgemv, 121
 - reproBLAS_dnrm2, 122
 - reproBLAS_dsum, 122
 - reproBLAS_dzasum, 123
 - reproBLAS_dznrm2, 123
 - reproBLAS_rcdotc_sub, 124
 - reproBLAS_rcdotu_sub, 124
 - reproBLAS_rcgemm, 124
 - reproBLAS_rcgemv, 125
 - reproBLAS_rcsum_sub, 127
 - reproBLAS_rdasum, 127
 - reproBLAS_rddot, 128
 - reproBLAS_rdgemm, 128
 - reproBLAS_rdgmv, 129
 - reproBLAS_rdnrm2, 130
 - reproBLAS_rdsum, 130
 - reproBLAS_rdzasum, 131
 - reproBLAS_rdznrm2, 131
 - reproBLAS_rsasum, 132
 - reproBLAS_rscasum, 132
 - reproBLAS_rscnrm2, 133
 - reproBLAS_rsdot, 133
 - reproBLAS_rsgemm, 134
 - reproBLAS_rsgmv, 135
 - reproBLAS_rsnrm2, 135
 - reproBLAS_rssum, 136
 - reproBLAS_rzdotc_sub, 136
 - reproBLAS_rzdotu_sub, 137
 - reproBLAS_rzgemm, 137
 - reproBLAS_rzgemv, 139
 - reproBLAS_rzsum_sub, 140
 - reproBLAS_sasum, 140
 - reproBLAS_scasum, 141
 - reproBLAS_scnrm2, 141
 - reproBLAS_sdot, 142
 - reproBLAS_sgemm, 142
 - reproBLAS_sgemv, 143
 - reproBLAS_snrm2, 144
 - reproBLAS_ssum, 144
 - reproBLAS_zdotc_sub, 145
 - reproBLAS_zdotu_sub, 145

reproBLAS_zgemm, 145
 reproBLAS_zgemv, 146
 reproBLAS_zsum_sub, 148
 reproBLAS_cdotc_sub
 reproBLAS.h, 116
 reproBLAS_cdotu_sub
 reproBLAS.h, 116
 reproBLAS_cgemm
 reproBLAS.h, 116
 reproBLAS_cgmv
 reproBLAS.h, 117
 reproBLAS_csum_sub
 reproBLAS.h, 118
 reproBLAS_dasum
 reproBLAS.h, 118
 reproBLAS_ddot
 reproBLAS.h, 119
 reproBLAS_dgemm
 reproBLAS.h, 119
 reproBLAS_dgemv
 reproBLAS.h, 121
 reproBLAS_dnrm2
 reproBLAS.h, 122
 reproBLAS_dsum
 reproBLAS.h, 122
 reproBLAS_dzasum
 reproBLAS.h, 123
 reproBLAS_dznrm2
 reproBLAS.h, 123
 reproBLAS_rcdotc_sub
 reproBLAS.h, 124
 reproBLAS_rcdotu_sub
 reproBLAS.h, 124
 reproBLAS_rcgemm
 reproBLAS.h, 124
 reproBLAS_rcgmv
 reproBLAS.h, 125
 reproBLAS_rcsum_sub
 reproBLAS.h, 127
 reproBLAS_rdasum
 reproBLAS.h, 127
 reproBLAS_rddot
 reproBLAS.h, 128
 reproBLAS_rdgemm
 reproBLAS.h, 128
 reproBLAS_rdgmv
 reproBLAS.h, 129
 reproBLAS_rdnrm2
 reproBLAS.h, 130
 reproBLAS_rdsun
 reproBLAS.h, 130
 reproBLAS_rdzasun
 reproBLAS.h, 131
 reproBLAS_rdznrm2
 reproBLAS.h, 131
 reproBLAS_rsasun
 reproBLAS.h, 132
 reproBLAS_rscasun
 reproBLAS.h, 132
 reproBLAS_rscnrm2
 reproBLAS.h, 133
 reproBLAS_rsdot
 reproBLAS.h, 133
 reproBLAS_rsgemm
 reproBLAS.h, 134
 reproBLAS_rsgmv
 reproBLAS.h, 135
 reproBLAS_rsnrm2
 reproBLAS.h, 135
 reproBLAS_rssun
 reproBLAS.h, 136
 reproBLAS_rzdotc_sub
 reproBLAS.h, 136
 reproBLAS_rzdotu_sub
 reproBLAS.h, 137
 reproBLAS_rzgemm
 reproBLAS.h, 137
 reproBLAS_rzgmv
 reproBLAS.h, 139
 reproBLAS_rzsum_sub
 reproBLAS.h, 140
 reproBLAS_sasun
 reproBLAS.h, 140
 reproBLAS_scasun
 reproBLAS.h, 141
 reproBLAS_scnrm2
 reproBLAS.h, 141
 reproBLAS_sdot
 reproBLAS.h, 142
 reproBLAS_sgemm
 reproBLAS.h, 142
 reproBLAS_sgmv
 reproBLAS.h, 143
 reproBLAS_snrm2
 reproBLAS.h, 144
 reproBLAS_ssum
 reproBLAS.h, 144
 reproBLAS_zdotc_sub
 reproBLAS.h, 145
 reproBLAS_zdotu_sub
 reproBLAS.h, 145
 reproBLAS_zgemm
 reproBLAS.h, 145
 reproBLAS_zgmv
 reproBLAS.h, 146
 reproBLAS_zsum_sub
 reproBLAS.h, 148
 SIWIDTH
 idxd.h, 14