

NVIDIA CUDA

Reference Manual

Version 3.0

February 2010

Contents

1	Dep	recated	List	1
2	Mod	lule Ind	ex	5
	2.1	Modul	es	5
3	Data	Struct	ure Index	7
	3.1	Data S	ructures	7
4	Mod	lule Do	umentation	9
	4.1	CUDA	Runtime API	9
		4.1.1	Detailed Description	9
	4.2	Thread	Management	10
		4.2.1	Detailed Description	10
		4.2.2	Function Documentation	10
			4.2.2.1 cudaThreadExit	10
			4.2.2.2 cudaThreadSynchronize	10
	4.3	Error l	landling	11
		4.3.1	Detailed Description	11
		4.3.2	Function Documentation	11
			4.3.2.1 cudaGetErrorString	11
			4.3.2.2 cudaGetLastError	11
	4.4	Device	Management	13
		4.4.1	Detailed Description	13
		4.4.2	Function Documentation	13
			4.4.2.1 cudaChooseDevice	13
			4.4.2.2 cudaGetDevice	14
			4.4.2.3 cudaGetDeviceCount	14
			4.4.2.4 cudaGetDeviceProperties	14
			4.4.2.5 cudaSetDevice	16

ii CONTENTS

		4.4.2.6	cudaSetDeviceFlags	16
		4.4.2.7	cudaSetValidDevices	17
4.5	Stream	n Managen	ment	19
	4.5.1	Detailed	Description	19
	4.5.2	Function	Documentation	19
		4.5.2.1	cudaStreamCreate	19
		4.5.2.2	cudaStreamDestroy	19
		4.5.2.3	cudaStreamQuery	20
		4.5.2.4	cudaStreamSynchronize	20
4.6	Event	Managem	ent	21
	4.6.1	Detailed	Description	21
	4.6.2	Function	Documentation	21
		4.6.2.1	cudaEventCreate	21
		4.6.2.2	cudaEventCreateWithFlags	22
		4.6.2.3	cudaEventDestroy	22
		4.6.2.4	cudaEventElapsedTime	22
		4.6.2.5	cudaEventQuery	23
		4.6.2.6	cudaEventRecord	23
		4.6.2.7	cudaEventSynchronize	24
4.7	Execu	tion Contro	ol	25
	4.7.1	Detailed	Description	25
	4.7.2	Function	Documentation	25
		4.7.2.1	cudaConfigureCall	25
		4.7.2.2	cudaFuncGetAttributes	26
		4.7.2.3	cudaFuncSetCacheConfig	26
		4.7.2.4	cudaLaunch	27
		4.7.2.5	cudaSetDoubleForDevice	27
		4.7.2.6	cudaSetDoubleForHost	28
		4.7.2.7	cudaSetupArgument	28
4.8	Memo	ry Manage	ement	29
	4.8.1	Detailed	Description	31
	4.8.2	Function	Documentation	32
		4.8.2.1	cudaFree	32
		4.8.2.2	cudaFreeArray	32
		4.8.2.3	cudaFreeHost	32
		4.8.2.4	cudaGetSymbolAddress	33
		4.8.2.5	cudaGetSymbolSize	33

	4.8.2.6	cudaHostAlloc	34
	4.8.2.7	cudaHostGetDevicePointer	35
	4.8.2.8	cudaHostGetFlags	35
	4.8.2.9	cudaMalloc	35
	4.8.2.10	cudaMalloc3D	36
	4.8.2.11	cudaMalloc3DArray	36
	4.8.2.12	cudaMallocArray	37
	4.8.2.13	cudaMallocHost	38
	4.8.2.14	cudaMallocPitch	39
	4.8.2.15	cudaMemcpy	39
	4.8.2.16	cudaMemcpy2D	40
	4.8.2.17	cudaMemcpy2DArrayToArray	41
	4.8.2.18	cudaMemcpy2DAsync	41
	4.8.2.19	cudaMemcpy2DFromArray	42
	4.8.2.20	cudaMemcpy2DFromArrayAsync	43
	4.8.2.21	cudaMemcpy2DToArray	44
	4.8.2.22	cudaMemcpy2DToArrayAsync	45
	4.8.2.23	cudaMemcpy3D	45
	4.8.2.24	cudaMemcpy3DAsync	47
	4.8.2.25	cudaMemcpyArrayToArray	48
	4.8.2.26	cudaMemcpyAsync	49
	4.8.2.27	cudaMemcpyFromArray	50
	4.8.2.28	cudaMemcpyFromArrayAsync	50
	4.8.2.29	cudaMemcpyFromSymbol	51
	4.8.2.30	cudaMemcpyFromSymbolAsync	52
	4.8.2.31	cudaMemcpyToArray	52
	4.8.2.32	cudaMemcpyToArrayAsync	53
	4.8.2.33	cudaMemcpyToSymbol	54
	4.8.2.34	cudaMemcpyToSymbolAsync	54
	4.8.2.35	cudaMemGetInfo	55
	4.8.2.36	cudaMemset	55
	4.8.2.37	cudaMemset2D	56
	4.8.2.38	cudaMemset3D	56
	4.8.2.39	make_cudaExtent	57
	4.8.2.40	make_cudaPitchedPtr	57
	4.8.2.41	make_cudaPos	58
4.9	OpenGL Interope	erability	59

iv CONTENTS

	4.9.1	Detailed	Description	59
	4.9.2	Function	Documentation	59
		4.9.2.1	cudaGLSetGLDevice	59
		4.9.2.2	cudaGraphicsGLRegisterBuffer	60
		4.9.2.3	cudaGraphicsGLRegisterImage	60
		4.9.2.4	cudaWGLGetDevice	61
4.10	OpenG	L Interope	erability [DEPRECATED]	62
	4.10.1	Detailed	Description	62
	4.10.2	Function	Documentation	62
		4.10.2.1	cudaGLMapBufferObject	62
		4.10.2.2	cudaGLMapBufferObjectAsync	63
		4.10.2.3	cudaGLRegisterBufferObject	63
		4.10.2.4	cudaGLSetBufferObjectMapFlags	64
		4.10.2.5	cudaGLUnmapBufferObject	65
		4.10.2.6	cudaGLUnmapBufferObjectAsync	65
		4.10.2.7	cudaGLUnregisterBufferObject	66
4.11	Direct3	3D 9 Interd	pperability	67
	4.11.1	Detailed	Description	67
	4.11.2	Function	Documentation	67
		4.11.2.1	cudaD3D9GetDevice	67
		4.11.2.2	cudaD3D9SetDirect3DDevice	68
		4.11.2.3	cudaGraphicsD3D9RegisterResource	68
4.12	Direct3		operability [DEPRECATED]	70
	4.12.1	Detailed	Description	70
	4.12.2	Function	Documentation	71
		4.12.2.1	cudaD3D9GetDirect3DDevice	71
		4.12.2.2	cudaD3D9MapResources	71
		4.12.2.3	cudaD3D9RegisterResource	72
		4.12.2.4	cudaD3D9ResourceGetMappedArray	73
		4.12.2.5	cudaD3D9ResourceGetMappedPitch	73
		4.12.2.6	cudaD3D9ResourceGetMappedPointer	74
		4.12.2.7	cudaD3D9ResourceGetMappedSize	75
		4.12.2.8	cudaD3D9ResourceGetSurfaceDimensions	76
		4.12.2.9	cudaD3D9ResourceSetMapFlags	76
			cudaD3D9UnmapResources	77
			cudaD3D9UnregisterResource	78
4.13	Direct3	3D 10 Inte	roperability	79

	4.13.1	Detailed Description	79
	4.13.2	Function Documentation	79
		4.13.2.1 cudaD3D10GetDevice	79
		4.13.2.2 cudaD3D10SetDirect3DDevice	80
		4.13.2.3 cudaGraphicsD3D10RegisterResource	80
4.14	Direct3	BD 10 Interoperability [DEPRECATED]	82
	4.14.1	Detailed Description	82
	4.14.2	Function Documentation	82
		4.14.2.1 cudaD3D10MapResources	82
		4.14.2.2 cudaD3D10RegisterResource	83
		4.14.2.3 cudaD3D10ResourceGetMappedArray	84
		4.14.2.4 cudaD3D10ResourceGetMappedPitch	85
		4.14.2.5 cudaD3D10ResourceGetMappedPointer	86
		4.14.2.6 cudaD3D10ResourceGetMappedSize	86
		4.14.2.7 cudaD3D10ResourceGetSurfaceDimensions	87
		4.14.2.8 cudaD3D10ResourceSetMapFlags	88
		4.14.2.9 cudaD3D10UnmapResources	88
		4.14.2.10 cudaD3D10UnregisterResource	89
4.15	Direct3	BD 11 Interoperability	90
	4.15.1	Detailed Description	90
	4.15.2	Function Documentation	90
		4.15.2.1 cudaD3D11GetDevice	90
		4.15.2.2 cudaD3D11SetDirect3DDevice	90
		4.15.2.3 cudaGraphicsD3D11RegisterResource	91
4.16	Graphi	cs Interoperability	93
	4.16.1	Detailed Description	93
	4.16.2	Function Documentation	93
		4.16.2.1 cudaGraphicsMapResources	93
		4.16.2.2 cudaGraphicsResourceGetMappedPointer	94
		4.16.2.3 cudaGraphicsResourceSetMapFlags	94
		4.16.2.4 cudaGraphicsSubResourceGetMappedArray	95
		4.16.2.5 cudaGraphicsUnmapResources	96
		4.16.2.6 cudaGraphicsUnregisterResource	96
4.17	Texture	e Reference Management	97
	4.17.1	Detailed Description	97
	4.17.2	Function Documentation	97
		4.17.2.1 cudaBindTexture	97

vi CONTENTS

99 99 100 101 101 102 102 102 103 104 104 104
100 101 101 102 102 102 103 104 104
100 101 101 102 102 102 103 104 104
101 101 102 102 102 103 104
101 102 102 102 103 104 104
102 102 102 102 103 104 104
102 102 102 103 104 104
102 102 103 104 104
102 103 104 104
103 104 104
104 104
104
104
105
105
106
106
107
107
108
108
109
109
110
110
111
112
115
115
115
115
115
115
115
115

CONTENTS vii

4.20.2.4 cudaFuncCache	17
4.20.2.5 cudaGraphicsCubeFace	.17
4.20.2.6 cudaGraphicsMapFlags	.17
4.20.2.7 cudaGraphicsRegisterFlags	.17
4.20.2.8 cudaMemcpyKind	.17
4.21 CUDA Driver API	18
4.21.1 Detailed Description	.18
4.22 Initialization	.19
4.22.1 Detailed Description	.19
4.22.2 Function Documentation	.19
4.22.2.1 cuInit	.19
4.23 Device Management	.20
4.23.1 Detailed Description	
4.23.2 Function Documentation	
4.23.2.1 cuDeviceComputeCapability	
4.23.2.2 cuDeviceGet	.21
4.23.2.3 cuDeviceGetAttribute	.21
4.23.2.4 cuDeviceGetCount	.23
4.23.2.5 cuDeviceGetName	.23
4.23.2.6 cuDeviceGetProperties	.23
4.23.2.7 cuDeviceTotalMem	.25
4.24 Version Management	.26
4.24.1 Detailed Description	.26
4.24.2 Function Documentation	.26
4.24.2.1 cuDriverGetVersion	.26
4.25 Context Management	.27
4.25.1 Detailed Description	.27
4.25.2 Function Documentation	27
4.25.2.1 cuCtxAttach	27
4.25.2.2 cuCtxCreate	28
4.25.2.3 cuCtxDestroy	29
4.25.2.4 cuCtxDetach	.29
4.25.2.5 cuCtxGetDevice	.30
4.25.2.6 cuCtxPopCurrent	.30
4.25.2.7 cuCtxPushCurrent	.31
4.25.2.8 cuCtxSynchronize	.31
4.26 Module Management	32

viii CONTENTS

	4.26.1	Detailed	Description
	4.26.2	Function	Documentation
		4.26.2.1	cuModuleGetFunction
		4.26.2.2	cuModuleGetGlobal
		4.26.2.3	cuModuleGetTexRef
		4.26.2.4	cuModuleLoad
		4.26.2.5	cuModuleLoadData
		4.26.2.6	cuModuleLoadDataEx
		4.26.2.7	cuModuleLoadFatBinary
		4.26.2.8	cuModuleUnload
4.27	Stream	Managem	nent
	4.27.1	Detailed	Description
	4.27.2	Function	Documentation
		4.27.2.1	cuStreamCreate
		4.27.2.2	cuStreamDestroy
		4.27.2.3	cuStreamQuery
		4.27.2.4	cuStreamSynchronize
4.28	Event I	Manageme	ent
	4.28.1	Detailed	Description
	4.28.2	Function	Documentation
		4.28.2.1	cuEventCreate
		4.28.2.2	cuEventDestroy
		4.28.2.3	cuEventElapsedTime
		4.28.2.4	cuEventQuery
		4.28.2.5	cuEventRecord
		4.28.2.6	cuEventSynchronize
4.29	Execut	ion Contro	ol
	4.29.1	Detailed	Description
	4.29.2	Function	Documentation
		4.29.2.1	cuFuncGetAttribute
		4.29.2.2	cuFuncSetBlockShape
		4.29.2.3	cuFuncSetCacheConfig
		4.29.2.4	cuFuncSetSharedSize
		4.29.2.5	cuLaunch
		4.29.2.6	cuLaunchGrid
		4.29.2.7	cuLaunchGridAsync
		4.29.2.8	cuParamSetf
		4.29.2.8	cuParamSetf

4.29.2.10 cuParamSetSize	
4.29.2.11 cuParamSetTexRef	. 150
4.29.2.12 cuParamSetv	. 150
4.30 Memory Management	. 152
4.30.1 Detailed Description	. 154
4.30.2 Function Documentation	. 154
4.30.2.1 cuArray3DCreate	. 154
4.30.2.2 cuArray3DGetDescriptor	. 156
4.30.2.3 cuArrayCreate	. 157
4.30.2.4 cuArrayDestroy	
4.30.2.5 cuArrayGetDescriptor	. 159
4.30.2.6 cuMemAlloc	. 159
4.30.2.7 cuMemAllocHost	. 160
4.30.2.8 cuMemAllocPitch	. 160
4.30.2.9 cuMemcpy2D	. 161
4.30.2.10 cuMemcpy2DAsync	. 163
4.30.2.11 cuMemcpy2DUnaligned	. 166
4.30.2.12 cuMemcpy3D	. 168
4.30.2.13 cuMemcpy3DAsync	. 170
4.30.2.14 cuMemcpyAtoA	. 172
4.30.2.15 cuMemcpyAtoD	. 173
4.30.2.16 cuMemcpyAtoH	. 173
4.30.2.17 cuMemcpyAtoHAsync	. 174
4.30.2.18 cuMemcpyDtoA	. 175
4.30.2.19 cuMemcpyDtoD	. 175
4.30.2.20 cuMemcpyDtoDAsync	. 176
4.30.2.21 cuMemcpyDtoH	. 176
4.30.2.22 cuMemcpyDtoHAsync	. 177
4.30.2.23 cuMemcpyHtoA	. 178
4.30.2.24 cuMemcpyHtoAAsync	. 178
4.30.2.25 cuMemcpyHtoD	. 179
4.30.2.26 cuMemcpyHtoDAsync	. 179
4.30.2.27 cuMemFree	. 180
4.30.2.28 cuMemFreeHost	. 181
4.30.2.29 cuMemGetAddressRange	. 181
4.30.2.30 cuMemGetInfo	. 182

		4.30.2.31	l cuMe	mHostA	lloc .	. 		 	 	 	 	 		182
		4.30.2.32	2 cuMe	mHostG	etDev	icePoi	inter .	 	 	 	 	 		183
		4.30.2.33	3 cuMe	mHostG	etFlag	,s		 	 	 	 	 		184
		4.30.2.34	4 cuMe	msetD1	6	. 		 	 	 	 	 		184
		4.30.2.35	5 cuMe	msetD2	D16 .	. 		 	 	 	 	 		185
		4.30.2.36	6 cuMe	msetD2	D32 .	. 		 	 	 	 	 		185
		4.30.2.37	7 cuMe	msetD2	D8	. 		 	 	 	 	 		186
		4.30.2.38	3 cuMe	msetD3	2	. 		 	 	 	 	 		187
		4.30.2.39	e cuMe	msetD8		. 		 	 	 	 	 		187
4.31	Texture	e Referenc	ce Mana	agement		. 		 	 	 	 	 		188
	4.31.1	Detailed 1	Descri	ption .		. 		 	 	 	 	 		189
	4.31.2	Function	Docun	nentatio	n	. 		 	 	 	 	 		189
		4.31.2.1	cuTex	RefCrea	ate	. 		 	 	 	 	 		189
		4.31.2.2	cuTex	RefDes	troy .	. 		 	 	 	 	 		189
		4.31.2.3	cuTex	RefGet	Addres	SS		 	 	 	 	 		189
		4.31.2.4	cuTex	RefGet	Addres	ssMod	le	 	 	 	 	 		190
		4.31.2.5	cuTex	RefGet	Array .	. 		 	 	 	 	 		190
		4.31.2.6	cuTex	RefGetl	Filter N	Iode .		 	 	 	 	 		191
		4.31.2.7	cuTex	RefGetl	Flags .	. 		 	 	 	 	 		191
		4.31.2.8	cuTex	RefGetl	Format	t		 	 	 	 	 		191
		4.31.2.9	cuTex	RefSet	Addres	s		 	 	 	 	 		192
		4.31.2.10	cuTex	RefSet	Addres	s2D .		 	 	 	 	 		192
		4.31.2.11	l cuTex	RefSet	Addres	sMode	e	 	 	 	 	 		193
		4.31.2.12	2 cuTex	RefSet	Array .	. 		 	 	 	 	 		193
		4.31.2.13	3 cuTex	RefSetF	∛ilterM	lode .		 	 	 	 	 		194
		4.31.2.14	4 cuTex	RefSetF	lags .	. 		 	 	 	 	 		194
		4.31.2.15	5 cuTex	RefSetF	⁷ ormat			 	 	 	 	 		195
4.32	OpenG	L Interope	erabilit	y		. 		 	 	 	 	 		196
	4.32.1	Detailed 1	Descri	ption .				 	 	 	 	 		196
	4.32.2	Function	Docun	nentatio	n			 	 	 	 	 		196
		4.32.2.1	cuGL	CtxCrea	ıte	. 		 	 	 	 	 		196
		4.32.2.2	cuGra	phicsGI	LRegis	terBu	ffer .	 	 	 	 	 		197
		4.32.2.3	cuGra	phicsGI	LRegis	terIma	age .	 	 	 	 	 		197
		4.32.2.4	cuWC	3LGetD	evice .			 	 	 	 	 		198
4.33	OpenG	L Interope	erabilit	y [DEPF	RECAT	ŒD]		 	 	 	 	 		199
	4.33.1	Detailed 1	Descrip	ption .				 	 	 	 	 		199
	4.33.2	Function	Docun	nentatio	n			 	 	 	 	 		199

	4.33.2.1	cuGLInit	 199
	4.33.2.2	cuGLMapBufferObject	 200
	4.33.2.3	cuGLMapBufferObjectAsync	 200
	4.33.2.4	cuGLRegisterBufferObject	 201
	4.33.2.5	cuGLSetBufferObjectMapFlags	 201
	4.33.2.6	cuGLUnmapBufferObject	 202
	4.33.2.7	cuGLUnmapBufferObjectAsync	 203
		cuGLUnregisterBufferObject	
4.34 Direct3	D 9 Interc	operability	 205
		Description	
		Documentation	
		cuD3D9CtxCreate	
		cuD3D9GetDevice	
		cuGraphicsD3D9RegisterResource	
		operability [DEPRECATED]	
		Description	
4.35.2	Function	Documentation	
	4.35.2.1		
	4.35.2.2	· · · · · · · · · · · · · · · · · · ·	
	4.35.2.3	cuD3D9RegisterResource	
	4.35.2.4	Transfer of the second	
		TER .	
	4.35.2.6	Tr.	
	4.35.2.7		
		cuD3D9ResourceGetSurfaceDimensions	
		cuD3D9ResourceSetMapFlags	
		cuD3D9UnmapResources	
		cuD3D9UnregisterResource	
		properability	
		Description	
		Documentation	
		cuD3D10CtxCreate	
		cuD3D10GetDevice	
		cuGraphicsD3D10RegisterResource	
		roperability [DEPRECATED]	
		Description	
4.37.2	Function	Documentation	 220

xii CONTENTS

	4.37.2.1	cuD3D10MapResources	220
	4.37.2.2	cuD3D10RegisterResource	221
	4.37.2.3	cuD3D10ResourceGetMappedArray	222
	4.37.2.4	cuD3D10ResourceGetMappedPitch	223
	4.37.2.5	cuD3D10ResourceGetMappedPointer	224
	4.37.2.6	cuD3D10ResourceGetMappedSize	225
	4.37.2.7	cuD3D10ResourceGetSurfaceDimensions	225
	4.37.2.8	cuD3D10ResourceSetMapFlags	226
	4.37.2.9	cuD3D10UnmapResources	227
	4.37.2.10	cuD3D10UnregisterResource	227
4.38 Direct3	3D 11 Inte	roperability	229
4.38.1	Detailed	Description	229
4.38.2	Function	Documentation	229
	4.38.2.1	cuD3D11CtxCreate	229
	4.38.2.2	cuD3D11GetDevice	230
	4.38.2.3	cuGraphicsD3D11RegisterResource	230
4.39 Graphi	cs Interop	erability	232
4.39.1	Detailed	Description	232
4.39.2	Function	Documentation	232
	4.39.2.1	cuGraphicsMapResources	232
	4.39.2.2	cuGraphicsResourceGetMappedPointer	233
	4.39.2.3	cuGraphicsResourceSetMapFlags	233
	4.39.2.4	cuGraphicsSubResourceGetMappedArray	234
	4.39.2.5	cuGraphicsUnmapResources	235
	4.39.2.6	cuGraphicsUnregisterResource	235
4.40 Data ty	pes used b	by CUDA driver	237
4.40.1	Define D	ocumentation	242
	4.40.1.1	CU_MEMHOSTALLOC_DEVICEMAP	242
	4.40.1.2	CU_MEMHOSTALLOC_PORTABLE	242
	4.40.1.3	CU_MEMHOSTALLOC_WRITECOMBINED	242
	4.40.1.4	CU_PARAM_TR_DEFAULT	242
	4.40.1.5	CU_TRSA_OVERRIDE_FORMAT	242
	4.40.1.6	CU_TRSF_NORMALIZED_COORDINATES	242
	4.40.1.7	CU_TRSF_READ_AS_INTEGER	242
	4.40.1.8	CUDA_VERSION	242
4.40.2	Typedef	Documentation	242
	4.40.2.1	CUaddress_mode	242

CONTENTS xiii

4.40.2.2	CUarray_cubemap_face	242
4.40.2.3	CUarray_format	243
4.40.2.4	CUcomputemode	243
4.40.2.5	CUctx_flags	243
4.40.2.6	CUDA_MEMCPY2D	243
4.40.2.7	CUDA_MEMCPY3D	243
4.40.2.8	CUdevice_attribute	243
4.40.2.9	CUdevprop	243
4.40.2.10	CUevent_flags	243
4.40.2.11	CUfilter_mode	243
4.40.2.12	CUfunc_cache	243
4.40.2.13	CUfunction_attribute	243
4.40.2.14	CUgraphicsMapResourceFlags	243
4.40.2.15	CUgraphicsRegisterFlags	244
4.40.2.16	CUjit_fallback	244
4.40.2.17	CUjit_option	244
4.40.2.18	CUjit_target	244
4.40.2.19	CUmemorytype	244
4.40.2.20	CUresult	244
Enumerat	ion Type Documentation	244
4.40.3.1	CUaddress_mode_enum	244
4.40.3.2	CUarray_cubemap_face_enum	244
4.40.3.3	CUarray_format_enum	245
4.40.3.4	CUcomputemode_enum	245
4.40.3.5	CUctx_flags_enum	245
4.40.3.6	cudaError_enum	245
4.40.3.7	CUdevice_attribute_enum	246
4.40.3.8	CUevent_flags_enum	247
4.40.3.9	CUfilter_mode_enum	248
4.40.3.10	CUfunc_cache_enum	248
4.40.3.11	CUfunction_attribute_enum	248
4.40.3.12	CUgraphicsMapResourceFlags_enum	248
4.40.3.13	CUgraphicsRegisterFlags_enum	248
4.40.3.14	CUjit_fallback_enum	248
4.40.3.15	CUjit_option_enum	249
4.40.3.16	CUjit_target_enum	250
4.40.3.17	CUmemorytype_enum	250
	4.40.2.3 4.40.2.4 4.40.2.5 4.40.2.6 4.40.2.7 4.40.2.8 4.40.2.10 4.40.2.11 4.40.2.12 4.40.2.13 4.40.2.14 4.40.2.15 4.40.2.17 4.40.2.17 4.40.2.18 4.40.2.19 4.40.2.20 Enumerat 4.40.3.1 4.40.3.2 4.40.3.3 4.40.3.4 4.40.3.5 4.40.3.8 4.40.3.7 4.40.3.8 4.40.3.11 4.40.3.12 4.40.3.13 4.40.3.11 4.40.3.15 4.40.3.15 4.40.3.15	4.40.2.2 CUarray_cubemap_face 4.40.2.3 CUarray_format 4.40.2.4 CUcomputemode 4.40.2.5 CUctx_flags 4.40.2.6 CUDA_MEMCPY2D 4.40.2.7 CUDA_MEMCPY3D 4.40.2.8 Cudevice_attribute 4.40.2.9 Cudevprop 4.40.2.10 Cuevent_flags 4.40.2.11 Cufinc_cache 4.40.2.12 Cufunc_cache 4.40.2.13 Cufunction_attribute 4.40.2.14 CugraphicsMapResourceFlags 4.40.2.15 CugraphicsRegisterFlags 4.40.2.16 Cujit_fallback 4.40.2.17 Cujit_option 4.40.2.18 Cujit_arget 4.40.2.19 Curemorytype 4.40.2.19 Curemorytype 4.40.2.19 Curemorytype 4.40.3.1 Culadfress_mode_enum 4.40.3.2 Cularray_cubemap_face_enum 4.40.3.3 Cuarray_cubemap_face_enum 4.40.3.4 Cuctx_flags_enum 4.40.3.5 Cutx_flags_enum 4.40.3.10 Cufunc_cache_enum 4.40.3.10 Cufunction_attribut_enum

5	Data	Structure Documentation	251
	5.1	CUDA_ARRAY3D_DESCRIPTOR Struct Reference	251
		5.1.1 Detailed Description	251
	5.2	CUDA_ARRAY_DESCRIPTOR Struct Reference	252
		5.2.1 Detailed Description	252
	5.3	CUDA_MEMCPY2D_st Struct Reference	253
		5.3.1 Detailed Description	254
	5.4	CUDA_MEMCPY3D_st Struct Reference	255
		5.4.1 Detailed Description	256
	5.5	cudaChannelFormatDesc Struct Reference	257
		5.5.1 Detailed Description	257
	5.6	cudaDeviceProp Struct Reference	258
		5.6.1 Detailed Description	259
	5.7	cudaExtent Struct Reference	260
		5.7.1 Detailed Description	260
	5.8	cudaFuncAttributes Struct Reference	261
		5.8.1 Detailed Description	261
	5.9	cudaMemcpy3DParms Struct Reference	262
		5.9.1 Detailed Description	262
	5.10	cudaPitchedPtr Struct Reference	263
		5.10.1 Detailed Description	263
	5.11	cudaPos Struct Reference	264
		5.11.1 Detailed Description	264
	5.12	CUdevprop_st Struct Reference	265
		5.12.1 Detailed Description	265

Chapter 1

Deprecated List

Global cudaGLMapBufferObject This function is deprecated as of Cuda 3.0.

Global cudaGLMapBufferObjectAsync This function is deprecated as of Cuda 3.0.

Global cudaGLRegisterBufferObject This function is deprecated as of Cuda 3.0.

Global cudaGLSetBufferObjectMapFlags This function is deprecated as of Cuda 3.0.

Global cudaGLUnmapBufferObject This function is deprecated as of Cuda 3.0.

Global cudaGLUnmapBufferObjectAsync This function is deprecated as of Cuda 3.0.

Global cudaGLUnregisterBufferObject This function is deprecated as of Cuda 3.0.

Global cudaD3D9GetDirect3DDevice This function is deprecated as of Cuda 3.0.

Global cudaD3D9MapResources This function is deprecated as of Cuda 3.0.

Global cudaD3D9RegisterResource This function is deprecated as of Cuda 3.0.

Global cudaD3D9ResourceGetMappedArray This function is deprecated as of Cuda 3.0.

Global cudaD3D9ResourceGetMappedPitch This function is deprecated as of Cuda 3.0.

Global cudaD3D9ResourceGetMappedPointer This function is deprecated as of Cuda 3.0.

2 Deprecated List

Global cudaD3D9ResourceGetMappedSize This function is deprecated as of Cuda 3.0.

Global cudaD3D9ResourceGetSurfaceDimensions This function is deprecated as of Cuda 3.0.

Global cudaD3D9ResourceSetMapFlags This function is deprecated as of Cuda 3.0.

Global cudaD3D9UnmapResources This function is deprecated as of Cuda 3.0.

Global cudaD3D9UnregisterResource This function is deprecated as of Cuda 3.0.

Global cudaD3D10MapResources This function is deprecated as of Cuda 3.0.

Global cudaD3D10RegisterResource This function is deprecated as of Cuda 3.0.

Global cudaD3D10ResourceGetMappedArray This function is deprecated as of Cuda 3.0.

Global cudaD3D10ResourceGetMappedPitch This function is deprecated as of Cuda 3.0.

Global cudaD3D10ResourceGetMappedPointer This function is deprecated as of Cuda 3.0.

Global cudaD3D10ResourceGetMappedSize This function is deprecated as of Cuda 3.0.

Global cudaD3D10ResourceGetSurfaceDimensions This function is deprecated as of Cuda 3.0.

Global cudaD3D10ResourceSetMapFlags This function is deprecated as of Cuda 3.0.

Global cudaD3D10UnmapResources This function is deprecated as of Cuda 3.0.

Global cudaD3D10UnregisterResource This function is deprecated as of Cuda 3.0.

Global cuGLInit This function is deprecated as of Cuda 3.0.

Global cuGLMapBufferObject This function is deprecated as of Cuda 3.0.

Global cuGLMapBufferObjectAsync This function is deprecated as of Cuda 3.0.

Global cuGLRegisterBufferObject This function is deprecated as of Cuda 3.0.

Global cuGLSetBufferObjectMapFlags This function is deprecated as of Cuda 3.0.

Global cuGLUnmapBufferObject This function is deprecated as of Cuda 3.0.

Global cuGLUnmapBufferObjectAsync This function is deprecated as of Cuda 3.0.

Global cuGLUnregisterBufferObject This function is deprecated as of Cuda 3.0.

Global cuD3D9GetDirect3DDevice This function is deprecated as of Cuda 3.0.

Global cuD3D9MapResources This function is deprecated as of Cuda 3.0.

Global cuD3D9RegisterResource This function is deprecated as of Cuda 3.0.

Global cuD3D9ResourceGetMappedArray This function is deprecated as of Cuda 3.0.

Global cuD3D9ResourceGetMappedPitch This function is deprecated as of Cuda 3.0.

Global cuD3D9ResourceGetMappedPointer This function is deprecated as of Cuda 3.0.

Global cuD3D9ResourceGetMappedSize This function is deprecated as of Cuda 3.0.

Global cuD3D9ResourceGetSurfaceDimensions This function is deprecated as of Cuda 3.0.

Global cuD3D9ResourceSetMapFlags This function is deprecated as of Cuda 3.0.

Global cuD3D9UnmapResources This function is deprecated as of Cuda 3.0.

Global cuD3D9UnregisterResource This function is deprecated as of Cuda 3.0.

Global cuD3D10MapResources This function is deprecated as of Cuda 3.0.

Global cuD3D10RegisterResource This function is deprecated as of Cuda 3.0.

4 Deprecated List

Global cuD3D10ResourceGetMappedArray This function is deprecated as of Cuda 3.0.

Global cuD3D10ResourceGetMappedPitch This function is deprecated as of Cuda 3.0.

Global cuD3D10ResourceGetMappedPointer This function is deprecated as of Cuda 3.0.

Global cuD3D10ResourceGetMappedSize This function is deprecated as of Cuda 3.0.

Global cuD3D10ResourceGetSurfaceDimensions This function is deprecated as of Cuda 3.0.

Global cuD3D10ResourceSetMapFlags This function is deprecated as of Cuda 3.0.

Global cuD3D10UnmapResources This function is deprecated as of Cuda 3.0.

Global cuD3D10UnregisterResource This function is deprecated as of Cuda 3.0.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

CUDA Runtime API	9
Thread Management	10
Error Handling	1
Device Management	13
Stream Management	19
Event Management	2
Execution Control	25
Memory Management	29
OpenGL Interoperability	59
OpenGL Interoperability [DEPRECATED]	62
Direct3D 9 Interoperability	6
Direct3D 9 Interoperability [DEPRECATED]	70
Direct3D 10 Interoperability	79
Direct3D 10 Interoperability [DEPRECATED]	82
Direct3D 11 Interoperability	90
Graphics Interoperability	93
Texture Reference Management	97
Version Management	102
C++ API Routines	103
Data types used by CUDA Runtime	112
CUDA Driver API	118
Initialization	119
Device Management	120
Version Management	
Context Management	
Module Management	132
Stream Management	138
Event Management	140
Execution Control	144
Memory Management	152
Texture Reference Management	188

 6 Module Index

Direct3D 9 Interoperability	205
Direct3D 9 Interoperability [DEPRECATED]	208
Direct3D 10 Interoperability	217
Direct3D 10 Interoperability [DEPRECATED]	220
Direct3D 11 Interoperability	229
Graphics Interoperability	232
Data types used by CUDA driver	237

Chapter 3

Data Structure Index

3.1 Data Structures

Here are	the d	lata struct	ures with	brief	descrip	tions
----------	-------	-------------	-----------	-------	---------	-------

CUDA_ARRAY3D_DESCRIPTOR
CUDA_ARRAY_DESCRIPTOR
CUDA_MEMCPY2D_st
CUDA_MEMCPY3D_st
cudaChannelFormatDesc
cudaDeviceProp
cudaExtent
cudaFuncAttributes
cudaMemcpy3DParms
cudaPitchedPtr
cudaPos
CUdevprop_st

8 Data Structure Index

Chapter 4

Module Documentation

4.1 CUDA Runtime API

Modules

- Thread Management
- · Error Handling
- Device Management
- Stream Management
- Event Management
- Execution Control
- Memory Management
- OpenGL Interoperability
- Direct3D 9 Interoperability
- Direct3D 10 Interoperability
- Direct3D 11 Interoperability
- Graphics Interoperability
- Texture Reference Management
- Version Management
- C++ API Routines

C++-style interface built on top of CUDA runtime API.

• Data types used by CUDA Runtime

4.1.1 Detailed Description

There are two levels for the runtime API.

The C API (cuda_runtime_api.h) is a C-style interface that does not require compiling with nvcc.

The C++ API (cuda_runtime.h) is a C++-style interface built on top of the C API. It wraps some of the C API routines, using overloading, references and default arguments. These wrappers can be used from C++ code and can be compiled with any C++ compiler. The C++ API also has some CUDA-specific wrappers that wrap C API routines that deal with symbols, textures, and device functions. These wrappers require the use of nvcc because they depend on code being generated by the compiler. For example, the execution configuration syntax to invoke kernels is only available in source code compiled with nvcc.

10 Module Documentation

4.2 Thread Management

Functions

cudaError_t cudaThreadExit (void)
 Exit and clean up from CUDA launches.

• cudaError_t cudaThreadSynchronize (void)

Wait for compute device to finish.

4.2.1 Detailed Description

This section describes the thread management functions of the CUDA runtime application programming interface.

4.2.2 Function Documentation

4.2.2.1 cudaError_t cudaThreadExit (void)

Explicitly cleans up all runtime-related resources associated with the calling host thread. Any subsequent API call reinitializes the runtime. cudaThreadExit() is implicitly called on host thread exit.

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaThreadSynchronize

4.2.2.2 cudaError_t cudaThreadSynchronize (void)

Blocks until the device has completed all preceding requested tasks. cudaThreadSynchronize() returns an error if one of the preceding tasks has failed. If the cudaDeviceBlockingSync flag was set for this device, the host thread will block until the device has finished its work.

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaThreadExit

4.3 Error Handling

4.3 Error Handling

Functions

const char * cudaGetErrorString (cudaError_t error)
 Returns the message string from an error code.

cudaError_t cudaGetLastError (void)

Returns the last error from a runtime call.

4.3.1 Detailed Description

This section describes the error handling functions of the CUDA runtime application programming interface.

4.3.2 Function Documentation

4.3.2.1 const char* cudaGetErrorString (cudaError t error)

Returns the message string from an error code.

Parameters:

error - Error code to convert to string

Returns:

char* pointer to a NULL-terminated string

See also:

cudaGetLastError, cudaError

4.3.2.2 cudaError_t cudaGetLastError (void)

Returns the last error that has been produced by any of the runtime calls in the same host thread and resets it to cudaSuccess.

Returns:

cudaSuccess, cudaErrorMissingConfiguration, cudaErrorMemoryAllocation, cudaErrorInitializationError, cudaErrorLaunchFailure, cudaErrorInvalidPitchValue cudaErrorInvalidConfiguration, cudaErrorInvalidDevice cudaErrorInvalidValue cudaErrorInvalidPitchValue cudaErrorInvalidSymbol cudaErrorInvalidDevicePointer cudaErrorInvalidTextureBinding cudaErrorInvalidChannelDescriptor cudaErrorInvalidMemcpyDirection cudaErrorAddressOfConstant cudaErrorTextureFetchFailed cudaErrorTextureNotBound cudaErrorSynchronizationError cudaErrorInvalidFilterSetting cudaErrorInvalidNormSetting cudaErrorMixedDeviceExecution cudaErrorCudartUnloading cudaErrorUnknown cudaErrorNotYetImplemented cudaErrorMemoryValueTooLarge cudaErrorInvalidResourceHandle cudaErrorNotReady cudaErrorInsufficient-Driver cudaErrorSetOnActiveProcess cudaErrorStartupFailure cudaErrorApiFailureBase

Module Documentation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetErrorString, cudaError

4.4 Device Management

Functions

- cudaError_t cudaChooseDevice (int *device, const struct cudaDeviceProp *prop)

 Select compute-device which best matches criteria.
- cudaError_t cudaGetDevice (int *device)
 Returns which device is currently being used.
- cudaError_t cudaGetDeviceCount (int *count)
 Returns the number of compute-capable devices.
- cudaError_t cudaGetDeviceProperties (struct cudaDeviceProp *prop, int device)

 Returns information about the compute-device.
- cudaError_t cudaSetDevice (int device)
 Set device to be used for GPU executions.
- cudaError_t cudaSetDeviceFlags (int flags)
 Sets flags to be used for device executions.
- cudaError_t cudaSetValidDevices (int *device_arr, int len)

 Set a list of devices that can be used for CUDA.

4.4.1 Detailed Description

This section describes the device management functions of the CUDA runtime application programming interface.

4.4.2 Function Documentation

4.4.2.1 cudaError_t cudaChooseDevice (int * device, const struct cudaDeviceProp * prop)

Returns in *device the device which has properties that best match *prop.

Parameters:

```
device - Device with best matchprop - Desired device properties
```

Returns:

cudaSuccess, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetDeviceCount, cudaGetDevice, cudaSetDevice, cudaGetDeviceProperties

14 Module Documentation

4.4.2.2 cudaError t cudaGetDevice (int * device)

Returns in *device the device on which the active host thread executes the device code.

Parameters:

device - Returns the device on which the active host thread executes the device code.

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetDeviceCount, cudaSetDevice, cudaGetDeviceProperties, cudaChooseDevice

4.4.2.3 cudaError_t cudaGetDeviceCount (int * count)

Returns in *count the number of devices with compute capability greater or equal to 1.0 that are available for execution. If there is no such device, cudaGetDeviceCount() returns 1 and device 0 only supports device emulation mode. Since this device will be able to emulate all hardware features, this device will report major and minor compute capability versions of 9999.

Parameters:

count - Returns the number of devices with compute capability greater or equal to 1.0

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetDevice, cudaSetDevice, cudaGetDeviceProperties, cudaChooseDevice

4.4.2.4 cudaError_t cudaGetDeviceProperties (struct cudaDeviceProp * prop, int device)

Returns in *prop the properties of device dev. The cudaDeviceProp structure is defined as:

```
struct cudaDeviceProp {
   char name[256];
   size_t totalGlobalMem;
   size_t sharedMemPerBlock;
   int regsPerBlock;
   int warpSize;
   size_t memPitch;
   int maxThreadsPerBlock;
   int maxThreadsDim[3];
   int maxGridSize[3];
```

```
size_t totalConstMem;
int major;
int minor;
int clockRate;
size_t textureAlignment;
int deviceOverlap;
int multiProcessorCount;
int kernelExecTimeoutEnabled;
int integrated;
int canMapHostMemory;
int computeMode;
int concurrentKernels;
}
```

where:

- name is an ASCII string identifying the device;
- totalGlobalMem is the total amount of global memory available on the device in bytes;
- sharedMemPerBlock is the maximum amount of shared memory available to a thread block in bytes; this amount is shared by all thread blocks simultaneously resident on a multiprocessor;
- regsPerBlock is the maximum number of 32-bit registers available to a thread block; this number is shared by all thread blocks simultaneously resident on a multiprocessor;
- warpSize is the warp size in threads;
- memPitch is the maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through cudaMallocPitch();
- maxThreadsPerBlock is the maximum number of threads per block;
- maxThreadsDim[3] contains the maximum size of each dimension of a block;
- maxGridSize[3] contains the maximum size of each dimension of a grid;
- clockRate is the clock frequency in kilohertz;
- totalConstMem is the total amount of constant memory available on the device in bytes;
- major, minor are the major and minor revision numbers defining the device's compute capability;
- textureAlignment is the alignment requirement; texture base addresses that are aligned to textureAlignment bytes do not need an offset applied to texture fetches;
- deviceOverlap is 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not;
- multiProcessorCount is the number of multiprocessors on the device;
- kernelExecTimeoutEnabled is 1 if there is a run time limit for kernels executed on the device, or 0 if not.
- integrated is 1 if the device is an integrated (motherboard) GPU and 0 if it is a discrete (card) component
- canMapHostMemory is 1 if the device can map host memory into the CUDA address space for use with cudaHostAlloc()/cudaHostGetDevicePointer(), or 0 if not;
- compute Mode is the compute mode that the device is currently in. Available modes are as follows:
 - cudaComputeModeDefault: Default mode Device is not restricted and multiple threads can use cudaSet-Device() with this device.

16 Module Documentation

cudaComputeModeExclusive: Compute-exclusive mode - Only one thread will be able to use cudaSetDevice() with this device.

- cudaComputeModeProhibited: Compute-prohibited mode No threads can use cudaSetDevice() with this
 device. Any errors from calling cudaSetDevice() with an exclusive (and occupied) or prohibited device will
 only show up after a non-device management runtime function is called. At that time, cudaErrorNoDevice
 will be returned.
- concurrentKernels is 1 if the device supports executing multiple kernels within the same context simultaneously, or 0 if not. It is not guaranteed that multiple kernels will be resident on the device concurrently so this feature should not be relied upon for correctness;

Parameters:

prop - Properties for the specified device

device - Device number to get properties for

Returns:

cudaSuccess, cudaErrorInvalidDevice

See also:

cudaGetDeviceCount, cudaGetDevice, cudaSetDevice, cudaChooseDevice

4.4.2.5 cudaError_t cudaSetDevice (int device)

Records device as the device on which the active host thread executes the device code. If the host thread has already initialized the CUDA runtime by calling non-device management runtime functions, this call returns cudaErrorSetOn-ActiveProcess.

Parameters:

device - Device on which the active host thread should execute the device code.

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorSetOnActiveProcess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetDeviceCount, cudaGetDevice, cudaGetDeviceProperties, cudaChooseDevice

4.4.2.6 cudaError_t cudaSetDeviceFlags (int flags)

Records flags as the flags to use when the active host thread executes device code. If the host thread has already initialized the CUDA runtime by calling non-device management runtime functions, this call returns cudaErrorSetOn-ActiveProcess.

The two LSBs of the flags parameter can be used to control how the CPU thread interacts with the OS scheduler when waiting for results from the device.

- cudaDeviceScheduleAuto: The default value if the flags parameter is zero, uses a heuristic based on the number of active CUDA contexts in the process C and the number of logical processors in the system P. If C > P, then CUDA will yield to other OS threads when waiting for the device, otherwise CUDA will not yield while waiting for results and actively spin on the processor.
- cudaDeviceScheduleSpin: Instruct CUDA to actively spin when waiting for results from the device. This can
 decrease latency when waiting for the device, but may lower the performance of CPU threads if they are performing work in parallel with the CUDA thread.
- cudaDeviceScheduleYield: Instruct CUDA to yield its thread when waiting for results from the device. This can increase latency when waiting for the device, but can increase the performance of CPU threads performing work in parallel with the device.
- cudaDeviceBlockingSync: Instruct CUDA to block the CPU thread on a synchronization primitive when waiting
 for the device to finish work.
- cudaDeviceMapHost: This flag must be set in order to allocate pinned host memory that is accessible to the device. If this flag is not set, cudaHostGetDevicePointer() will always return a failure code.
- cudaDeviceLmemResizeToMax: Instruct CUDA to not reduce local memory after resizing local memory for a kernel. This can prevent thrashing by local memory allocations when launching many kernels with high local memory usage at the cost of potentially increased memory usage.

Parameters:

flags - Parameters for device operation

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorSetOnActiveProcess

See also:

cudaGetDeviceCount, cudaGetDevice, cudaGetDeviceProperties, cudaSetDevice, cudaSetValidDevices, cudaChooseDevice

4.4.2.7 cudaError_t cudaSetValidDevices (int * device_arr, int len)

Sets a list of devices for CUDA execution in priority order using device_arr. The parameter len specifies the number of elements in the list. CUDA will try devices from the list sequentially until it finds one that works. If this function is not called, or if it is called with a len of 0, then CUDA will go back to its default behavior of trying devices sequentially from a default list containing all of the available CUDA devices in the system. If a specified device ID in the list does not exist, this function will return cudaErrorInvalidDevice. If len is not 0 and device_arr is NULL or if len is greater than the number of devices in the system, then cudaErrorInvalidValue is returned.

Parameters:

```
device_arr - List of devices to trylen - Number of devices in specified list
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevice

Note:

Note that this function may also return error codes from previous, asynchronous launches.

18 Module Documentation

See	പ	مما	
266	ж	SO	١.

cudaGetDeviceCount, cudaSetDevice, cudaGetDeviceProperties, cudaSetDeviceFlags, cudaChooseDevice

4.5 Stream Management

Functions

• cudaError_t cudaStreamCreate (cudaStream_t *pStream)

Create an asynchronous stream.

• cudaError_t cudaStreamDestroy (cudaStream_t stream)

Destroys and cleans up an asynchronous stream.

cudaError_t cudaStreamQuery (cudaStream_t stream)

Queries an asynchronous stream for completion status.

• cudaError_t cudaStreamSynchronize (cudaStream_t stream)

Waits for stream tasks to complete.

4.5.1 Detailed Description

This section describes the stream management functions of the CUDA runtime application programming interface.

4.5.2 Function Documentation

4.5.2.1 cudaError_t cudaStreamCreate (cudaStream_t * pStream)

Creates a new asynchronous stream.

Parameters:

pStream - Pointer to new stream identifier

Returns:

cudaSuccess, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaStreamQuery, cudaStreamSynchronize, cudaStreamDestroy

4.5.2.2 cudaError_t cudaStreamDestroy (cudaStream_t stream)

Destroys and cleans up the asynchronous stream specified by stream.

Parameters:

stream - Stream identifier

20 Module Documentation

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaStreamCreate, cudaStreamQuery, cudaStreamSynchronize

4.5.2.3 cudaError_t cudaStreamQuery (cudaStream_t stream)

Returns cudaSuccess if all operations in stream have completed, or cudaErrorNotReady if not.

Parameters:

stream - Stream identifier

Returns:

cudaSuccess, cudaErrorNotReady cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaStreamCreate, cudaStreamSynchronize, cudaStreamDestroy

4.5.2.4 cudaError_t cudaStreamSynchronize (cudaStream_t stream)

Blocks until stream has completed all operations. If the cudaDeviceBlockingSync flag was set for this device, the host thread will block until the stream is finished with all of its tasks.

Parameters:

stream - Stream identifier

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaStreamCreate, cudaStreamQuery, cudaStreamDestroy

4.6 Event Management 21

4.6 Event Management

Functions

cudaError_t cudaEventCreate (cudaEvent_t *event)

Creates an event object.

cudaError_t cudaEventCreateWithFlags (cudaEvent_t *event, int flags)

Creates an event object with the specified flags.

cudaError t cudaEventDestroy (cudaEvent t event)

Destroys an event object.

• cudaError_t cudaEventElapsedTime (float *ms, cudaEvent_t start, cudaEvent_t end)

Computes the elapsed time between events.

cudaError_t cudaEventQuery (cudaEvent_t event)

Query if an event has been recorded.

• cudaError_t cudaEventRecord (cudaEvent_t event, cudaStream_t stream)

Records an event.

• cudaError_t cudaEventSynchronize (cudaEvent_t event)

Wait for an event to be recorded.

4.6.1 Detailed Description

This section describes the event management functions of the CUDA runtime application programming interface.

4.6.2 Function Documentation

4.6.2.1 cudaError_t cudaEventCreate (cudaEvent_t * event)

Creates an event object using cudaEventDefault.

Parameters:

event - Newly created event

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalidValue, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda Event Create With Flags, \ cuda Event Record, \ cuda Event Query, \ cuda Event Synchronize, \ cuda Event Destroy, \ cuda Event Elapsed Time$

4.6.2.2 cudaError_t cudaEventCreateWithFlags (cudaEvent_t * event, int flags)

Creates an event object with the specified flags. Valid flags include:

- cudaEventDefault: Default event creation flag.
- cudaEventBlockingSync: Specifies that event should use blocking synchronization. A host thread that uses cudaEventSynchronize() to wait on an event created with this flag will block until the event actually completes.

Parameters:

```
event - Newly created eventflags - Flags for new event
```

Returns:

 $cuda Success, \quad cuda Error Initialization Error, \quad cuda Error Prior Launch Failure, \quad cuda Error Invalid Value, \quad cuda Error Memory Allocation$

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda Event Create, \ cuda Event Record, \ cuda Event Query, \ cuda Event Synchronize, \ cuda Event Destroy, \ cuda Event Eve$

4.6.2.3 cudaError_t cudaEventDestroy (cudaEvent_t event)

Destroys the specified event object.

Parameters:

```
event - Event to destroy
```

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Event Create, cuda Event Create With Flags, cuda Event Query, cuda Event Synchronize, cuda Event Record, cuda Event Elapsed Time

4.6.2.4 cudaEvent_t cudaEvent_t end) cudaEvent_t start, cudaEvent_t end)

Computes the elapsed time between two events (in milliseconds with a resolution of around 0.5 microseconds). If either event has not been recorded yet, this function returns cudaErrorInvalidValue. If either event has been recorded with a non-zero stream, the result is undefined.

4.6 Event Management 23

Parameters:

```
ms - Time between start and stop in msstart - Starting eventend - Stopping event
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda Event Create With Flags, \ cuda Event Query, \ cuda Event Synchronize, \ cuda Event Destroy, \ cuda Event Record$

4.6.2.5 cudaError_t cudaEventQuery (cudaEvent_t event)

Returns cudaSuccess if the event has actually been recorded, or cudaErrorNotReady if not. If cudaEventRecord() has not been called on this event, the function returns cudaErrorInvalidValue.

Parameters:

```
event - Event to query
```

Returns:

cudaSuccess, cudaErrorNotReady, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalid-Value cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Event Create, cuda Event Create With Flags, cuda Event Record, cuda Event Synchronize, cuda Event Destroy, cuda Event Elapsed Time

4.6.2.6 cudaError_t cudaEventRecord (cudaEvent_t event, cudaStream_t stream)

Records an event. If stream is non-zero, the event is recorded after all preceding operations in the stream have been completed; otherwise, it is recorded after all preceding operations in the CUDA context have been completed. Since this operation is asynchronous, cudaEventQuery() and/or cudaEventSynchronize() must be used to determine when the event has actually been recorded.

If cudaEventRecord() has previously been called and the event has not been recorded yet, this function returns cudaErrorInvalidValue.

Parameters:

event - Event to record

stream - Stream in which to record event

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Event Create, cuda Event Create With Flags, cuda Event Query, cuda Event Synchronize, cuda Event Destroy, cuda Event Elapsed Time

4.6.2.7 cudaError_t cudaEventSynchronize (cudaEvent_t event)

Blocks until the event has actually been recorded. If cudaEventRecord() has not been called on this event, the function returns cudaErrorInvalidValue. Waiting for an event that was created with the cudaEventBlockingSync flag will cause the calling host thread to block until the event has actually been recorded.

Parameters:

event - Event to wait for

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalidValue cudaErrorInvalidResourceHandle

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda Event Create, \ cuda Event Create With Flags, \ cuda Event Record, \ cuda Event Query, \ cuda Event Destroy, \ cuda Event Elapsed Time$

4.7 Execution Control 25

4.7 Execution Control

Functions

cudaError_t cudaConfigureCall (dim3 gridDim, dim3 blockDim, size_t sharedMem, cudaStream_t stream)
 Configure a device-launch.

• cudaError_t cudaFuncGetAttributes (struct cudaFuncAttributes *attr, const char *func) Find out attributes for a given function.

• cudaError_t cudaFuncSetCacheConfig (const char *func, enum cudaFuncCache cacheConfig)

Sets the preferred cache configuration for a device function.

cudaError_t cudaLaunch (const char *entry)
 Launches a device function.

cudaError_t cudaSetDoubleForDevice (double *d)
 Converts a double argument to be executed on a device.

cudaError_t cudaSetDoubleForHost (double *d)
 Converts a double argument after execution on a device.

• cudaError_t cudaSetupArgument (const void *arg, size_t size, size_t offset)

Configure a device launch.

4.7.1 Detailed Description

This section describes the execution control functions of the CUDA runtime application programming interface.

4.7.2 Function Documentation

4.7.2.1 cudaError_t cudaConfigureCall (dim3 gridDim, dim3 blockDim, size_t sharedMem, cudaStream_t stream)

Specifies the grid and block dimensions for the device call to be executed similar to the execution configuration syntax. cudaConfigureCall() is stack based. Each call pushes data on top of an execution stack. This data contains the dimension for the grid and thread blocks, together with any arguments for the call.

Parameters:

```
gridDim - Grid dimensionsblockDim - Block dimensionssharedMem - Shared memorystream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidConfiguration

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C API),

4.7.2.2 cudaError_t cudaFuncGetAttributes (struct cudaFuncAttributes * attr, const char * func)

This function obtains the attributes of a function specified via func, which is a character string that specifies the fully-decorated (C++) name for a function that executes on the device. The parameter specified by func must be declared as a __global__ function. The fetched attributes are placed in attr. If the specified function does not exist, then cudaErrorInvalidDeviceFunction is returned.

Parameters:

```
attr - Return pointer to function's attributesfunc - Function to get attributes of
```

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorInvalidDeviceFunction

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C++ API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C API)

4.7.2.3 cudaError_t cudaFuncSetCacheConfig (const char * func, enum cudaFuncCache cacheConfig)

On devices where the L1 cache and shared memory use the same hardware resources, this sets through cacheConfig the preferred cache configuration for the function specified via func. This is only a preference. The runtime will use the requested configuration if possible, but it is free to choose a different configuration if required to execute func.

func is a character string that specifies the fully-decorated (C++) name for a function that executes on the device. The parameter specified by func must be declared as a __global__ function. If the specified function does not exist, then cudaErrorInvalidDeviceFunction is returned.

This setting does nothing on devices where the size of the L1 cache and shared memory are fixed.

Switching between configuration modes may insert a device-side synchronization point for streamed kernel launches.

Parameters:

```
func - Device char string naming device functioncacheConfig - Cache configuration mode
```

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorInvalidDeviceFunction

4.7 Execution Control 27

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C++ API), cudaFuncGetAttributes (C API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C API)

4.7.2.4 cudaError_t cudaLaunch (const char * entry)

Launches the function entry on the device. The parameter entry must be a character string naming a function that executes on the device. The parameter specified by entry must be declared as a __global__ function. cudaLaunch() must be preceded by a call to cudaConfigureCall() since it pops the data that was pushed by cudaConfigureCall() from the execution stack.

Parameters:

entry - Device char string naming device function to execute

Returns:

cudaSuccess, cudaErrorInvalidDeviceFunction, cudaErrorInvalidConfiguration, cudaErrorLaunchFailure, cudaErrorLaunchOutOfResources

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C API), cudaLaunch (C++ API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C API)

4.7.2.5 cudaError_t cudaSetDoubleForDevice (double * d)

Parameters:

d - Double to convert

Converts the double value of d to an internal float representation if the device does not support double arithmetic. If the device does natively support doubles, then this function does nothing.

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C API), cudaLaunch (C API), cudaSetDoubleForHost, cudaSetupArgument (C API)

4.7.2.6 cudaError t cudaSetDoubleForHost (double * d)

Converts the double value of d from a potentially internal float representation if the device does not support double arithmetic. If the device does natively support doubles, then this function does nothing.

Parameters:

d - Double to convert

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetupArgument (C API)

4.7.2.7 cudaError_t cudaSetupArgument (const void * arg, size_t size, size_t offset)

Pushes size bytes of the argument pointed to by arg at offset bytes from the start of the parameter passing area, which starts at offset 0. The arguments are stored in the top of the execution stack. cudaSetupArgument() must be preceded by a call to cudaConfigureCall().

Parameters:

```
arg - Argument to push for a kernel launchsize - Size of argumentoffset - Offset in argument stack to push new arg
```

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C++ API),

4.8 Memory Management

Functions

• cudaError_t cudaFree (void *devPtr)

Frees memory on the device.

cudaError_t cudaFreeArray (struct cudaArray *array)

Frees an array on the device.

cudaError_t cudaFreeHost (void *ptr)

Frees page-locked memory.

• cudaError_t cudaGetSymbolAddress (void **devPtr, const char *symbol)

Finds the address associated with a CUDA symbol.

• cudaError t cudaGetSymbolSize (size t *size, const char *symbol)

Finds the size of the object associated with a CUDA symbol.

cudaError_t cudaHostAlloc (void **ptr, size_t size, unsigned int flags)

Allocates page-locked memory on the host.

cudaError_t cudaHostGetDevicePointer (void **pDevice, void *pHost, unsigned int flags)

Passes back device pointer of mapped host memory allocated by cudaHostAlloc().

• cudaError_t cudaHostGetFlags (unsigned int *pFlags, void *pHost)

Passes back flags used to allocate pinned host memory allocated by cudaHostAlloc().

• cudaError t cudaMalloc (void **devPtr, size t size)

Allocate memory on the device.

cudaError t cudaMalloc3D (struct cudaPitchedPtr *pitchedDevPtr, struct cudaExtent extent)

Allocates logical 1D, 2D, or 3D memory objects on the device.

• cudaError_t cudaMalloc3DArray (struct cudaArray **arrayPtr, const struct cudaChannelFormatDesc *desc, struct cudaExtent extent)

Allocate an array on the device.

cudaError_t cudaMallocArray (struct cudaArray **arrayPtr, const struct cudaChannelFormatDesc *desc, size_t width, size t height)

Allocate an array on the device.

cudaError_t cudaMallocHost (void **ptr, size_t size)

Allocates page-locked memory on the host.

• cudaError_t cudaMallocPitch (void **devPtr, size_t *pitch, size_t width, size_t height)

Allocates pitched memory on the device.

• cudaError t cudaMemcpy (void *dst, const void *src, size t count, enum cudaMemcpyKind kind)

Copies data between host and device.

• cudaError_t cudaMemcpy2D (void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies data between host and device.

 cudaError_t cudaMemcpy2DArrayToArray (struct cudaArray *dst, size_t wOffsetDst, size_t hOffsetDst, const struct cudaArray *src, size_t wOffsetSrc, size_t hOffsetSrc, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies data between host and device.

cudaError_t cudaMemcpy2DAsync (void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data between host and device.

• cudaError_t cudaMemcpy2DFromArray (void *dst, size_t dpitch, const struct cudaArray *src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies data between host and device.

cudaError_t cudaMemcpy2DFromArrayAsync (void *dst, size_t dpitch, const struct cudaArray *src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data between host and device.

• cudaError_t cudaMemcpy2DToArray (struct cudaArray *dst, size_t wOffset, size_t hOffset, const void *src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies data between host and device.

• cudaError_t cudaMemcpy2DToArrayAsync (struct cudaArray *dst, size_t wOffset, size_t hOffset, const void *src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data between host and device.

cudaError_t cudaMemcpy3D (const struct cudaMemcpy3DParms *p)

Copies data between 3D objects.

• cudaError_t cudaMemcpy3DAsync (const struct cudaMemcpy3DParms *p, cudaStream_t stream)

Copies data between 3D objects.

• cudaError_t cudaMemcpyArrayToArray (struct cudaArray *dst, size_t wOffsetDst, size_t hOffsetDst, const struct cudaArray *src, size_t wOffsetSrc, size_t count, enum cudaMemcpyKind kind)

Copies data between host and device.

cudaError_t cudaMemcpyAsync (void *dst, const void *src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data between host and device.

• cudaError_t cudaMemcpyFromArray (void *dst, const struct cudaArray *src, size_t wOffset, size_t hOffset, size_t count, enum cudaMemcpyKind kind)

Copies data between host and device.

• cudaError_t cudaMemcpyFromArrayAsync (void *dst, const struct cudaArray *src, size_t wOffset, size_t hOffset, size t count, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data between host and device.

cudaError_t cudaMemcpyFromSymbol (void *dst, const char *symbol, size_t count, size_t offset, enum cud-aMemcpyKind kind)

Copies data from the given symbol on the device.

• cudaError_t cudaMemcpyFromSymbolAsync (void *dst, const char *symbol, size_t count, size_t offset, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data from the given symbol on the device.

cudaError_t cudaMemcpyToArray (struct cudaArray *dst, size_t wOffset, size_t hOffset, const void *src, size_t count, enum cudaMemcpyKind kind)

Copies data between host and device.

• cudaError_t cudaMemcpyToArrayAsync (struct cudaArray *dst, size_t wOffset, size_t hOffset, const void *src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data between host and device.

cudaError_t cudaMemcpyToSymbol (const char *symbol, const void *src, size_t count, size_t offset, enum cudaMemcpyKind kind)

Copies data to the given symbol on the device.

• cudaError_t cudaMemcpyToSymbolAsync (const char *symbol, const void *src, size_t count, size_t offset, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies data to the given symbol on the device.

cudaError_t cudaMemGetInfo (size_t *free, size_t *total)

Gets free and total device memory.

• cudaError_t cudaMemset (void *devPtr, int value, size_t count)

Initializes or sets device memory to a value.

cudaError_t cudaMemset2D (void *devPtr, size_t pitch, int value, size_t width, size_t height)

Initializes or sets device memory to a value.

• cudaError_t cudaMemset3D (struct cudaPitchedPtr pitchedDevPtr, int value, struct cudaExtent extent)

Initializes or sets device memory to a value.

• struct cudaExtent make_cudaExtent (size_t w, size_t h, size_t d)

Returns a cudaExtent based on input parameters.

• struct cudaPitchedPtr make_cudaPitchedPtr (void *d, size_t p, size_t xsz, size_t ysz)

Returns a cudaPitchedPtr based on input parameters.

• struct cudaPos make_cudaPos (size_t x, size_t y, size_t z)

Returns a cudaPos based on input parameters.

4.8.1 Detailed Description

This section describes the memory management functions of the CUDA runtime application programming interface.

4.8.2 Function Documentation

4.8.2.1 cudaError_t cudaFree (void * devPtr)

Frees the memory space pointed to by devPtr, which must have been returned by a previous call to cudaMalloc() or cudaMallocPitch(). Otherwise, or if cudaFree(devPtr) has already been called before, an error is returned. If devPtr is 0, no operation is performed. cudaFree() returns cudaErrorInvalidDevicePointer in case of failure.

Parameters:

devPtr - Device pointer to memory to free

Returns:

cudaSuccess, cudaErrorInvalidDevicePointer, cudaErrorInitializationError

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc, cudaMallocPitch, cudaMallocArray, cudaFreeArray, cudaMallocHost, cudaFreeHost, cudaMalloc3D, cudaMalloc3DArray, cudaHostAlloc

4.8.2.2 cudaError_t cudaFreeArray (struct cudaArray * array)

Frees the CUDA array array, which must have been * returned by a previous call to cudaMallocArray(). If cudaFreeArray(array) has already been called before, cudaErrorInvalidValue is returned. If devPtr is 0, no operation is performed.

Parameters:

array - Pointer to array to free

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInitializationError

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc, cudaMallocPitch, cudaFree, cudaMallocArray, cudaMallocHost, cudaFreeHost, cudaHostAlloc

4.8.2.3 cudaError_t cudaFreeHost (void * ptr)

Frees the memory space pointed to by hostPtr, which must have been returned by a previous call to cudaMallocHost() or cudaHostAlloc().

Parameters:

ptr - Pointer to memory to free

Returns:

cudaSuccess, cudaErrorInitializationError

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc, cudaMallocPitch, cudaFree, cudaMallocArray, cudaFreeArray, cudaMallocHost, cudaMalloc3D, cudaMalloc3DArray, cudaHostAlloc

4.8.2.4 cudaError_t cudaGetSymbolAddress (void ** devPtr, const char * symbol)

Returns in *devPtr the address of symbol symbol on the device. symbol can either be a variable that resides in global memory space, or it can be a character string, naming a variable that resides in global memory space. If symbol cannot be found, or if symbol is not declared in the global memory space, *devPtr is unchanged and the error cudaErrorInvalidSymbol is returned.

Parameters:

```
devPtr - Return device pointer associated with symbolsymbol - Global variable or string symbol to search for
```

Returns:

cudaSuccess, cudaErrorInvalidSymbol, cudaErrorAddressOfConstant

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetSymbolAddress (C++ API) cudaGetSymbolSize (C API)

4.8.2.5 cudaError_t cudaGetSymbolSize (size_t * size, const char * symbol)

Returns in *size the size of symbol symbol. symbol can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. If symbol cannot be found, or if symbol is not declared in global or constant memory space, *size is unchanged and the error cudaErrorInvalidSymbol is returned.

Parameters:

```
size - Size of object associated with symbolsymbol - Global variable or string symbol to find size of
```

Returns:

cudaSuccess, cudaErrorInvalidSymbol

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetSymbolAddress (C API) cudaGetSymbolSize (C++ API)

4.8.2.6 cudaError_t cudaHostAlloc (void ** ptr, size_t size, unsigned int flags)

Allocates size bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as cudaMemcpy(). Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as malloc(). Allocating excessive amounts of pinned memory may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

The flags parameter enables different options to be specified that affect the allocation, as follows.

- cudaHostAllocDefault: This flag's value is defined to be 0 and causes cudaHostAlloc() to emulate cudaMallocHost().
- cudaHostAllocPortable: The memory returned by this call will be considered as pinned memory by all CUDA
 contexts, not just the one that performed the allocation.
- cudaHostAllocMapped: Maps the allocation into the CUDA address space. The device pointer to the memory may be obtained by calling cudaHostGetDevicePointer().
- cudaHostAllocWriteCombined: Allocates the memory as write-combined (WC). WC memory can be transferred across the PCI Express bus more quickly on some system configurations, but cannot be read efficiently by most CPUs. WC memory is a good option for buffers that will be written by the CPU and read by the device via mapped pinned memory or host->device transfers.

All of these flags are orthogonal to one another: a developer may allocate memory that is portable, mapped and/or write-combined with no restrictions.

cudaSetDeviceFlags() must have been called with the cudaDeviceMapHost flag in order for the cudaHostAllocMapped flag to have any effect.

The cudaHostAllocMapped flag may be specified on CUDA contexts for devices that do not support mapped pinned memory. The failure is deferred to cudaHostGetDevicePointer() because the memory may be mapped into other CUDA contexts via the cudaHostAllocPortable flag.

Memory allocated by this function must be freed with cudaFreeHost().

Parameters:

ptr - Device pointer to allocated memory

size - Requested allocation size in bytes

flags - Requested properties of allocated memory

Returns:

cudaSuccess, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaSetDeviceFlags, cudaMallocHost, cudaFreeHost

4.8.2.7 cudaError_t cudaHostGetDevicePointer (void ** pDevice, void * pHost, unsigned int flags)

Passes back the device pointer corresponding to the mapped, pinned host buffer allocated by cudaHostAlloc().

cudaHostGetDevicePointer() will fail if the cudaDeviceMapHost flag was not specified before deferred context creation occurred, or if called on a device that does not support mapped, pinned memory.

flags provides for future releases. For now, it must be set to 0.

Parameters:

```
pDevice - Returned device pointer for mapped memorypHost - Requested host pointer mappingflags - Flags for extensions (must be 0 for now)
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaSetDeviceFlags, cudaHostAlloc

4.8.2.8 cudaError_t cudaHostGetFlags (unsigned int * pFlags, void * pHost)

cudaHostGetFlags() will fail if the input pointer does not reside in an address range allocated by cudaHostAlloc().

Parameters:

```
pFlags - Returned flags wordpHost - Host pointer
```

Returns:

cudaSuccess, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaHostAlloc

4.8.2.9 cudaError_t cudaMalloc (void ** devPtr, size_t size)

Allocates size bytes of linear memory on the device and returns in *devPtr a pointer to the allocated memory. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. cudaMalloc() returns cudaErrorMemoryAllocation in case of failure.

Parameters:

devPtr - Pointer to allocated device memorysize - Requested allocation size in bytes

Returns:

cudaSuccess, cudaErrorMemoryAllocation

See also:

cudaMallocPitch, cudaFree, cudaMallocArray, cudaFreeArray, cudaMalloc3D, cudaMalloc3DArray, cudaMallocHost, cudaFreeHost, cudaHostAlloc

4.8.2.10 cudaError_t cudaMalloc3D (struct cudaPitchedPtr * pitchedDevPtr, struct cudaExtent extent)

Allocates at least width * height * depth bytes of linear memory on the device and returns a cudaPitchedPtr in which ptr is a pointer to the allocated memory. The function may pad the allocation to ensure hardware alignment requirements are met. The pitch returned in the pitch field of pitchedDevPtr is the width in bytes of the allocation.

The returned cudaPitchedPtr contains additional fields xsize and ysize, the logical width and height of the allocation, which are equivalent to the width and height extent parameters provided by the programmer during allocation.

For allocations of 2D and 3D objects, it is highly recommended that programmers perform allocations using cudaMalloc3D() or cudaMallocPitch(). Due to alignment restrictions in the hardware, this is especially true if the application will be performing memory copies involving 2D or 3D objects (whether linear memory or CUDA arrays).

Parameters:

```
pitchedDevPtr - Pointer to allocated pitched device memoryextent - Requested allocation size
```

Returns:

cudaSuccess, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMallocPitch, cudaFree, cudaMemcpy3D, cudaMemset3D, cudaMalloc3DArray, cudaMallocArray, cudaFreeArray, cudaMallocHost, cudaFreeHost, cudaHostAlloc, make_cudaPitchedPtr, make_cudaExtent

4.8.2.11 cudaError_t cudaMalloc3DArray (struct cudaArray ** arrayPtr, const struct cudaChannelFormatDesc * desc, struct cudaExtent extent)

Allocates a CUDA array according to the cudaChannelFormatDesc structure desc and returns a handle to the new CUDA array in *arrayPtr.

The cudaChannelFormatDesc is defined as:

```
struct cudaChannelFormatDesc {
   int x, y, z, w;
   enum cudaChannelFormatKind f;
};
```

where cudaChannelFormatKind is one of cudaChannelFormatKindSigned, cudaChannelFormatKindUnsigned, or cudaChannelFormatKindFloat.

cudaMalloc3DArray() is able to allocate 1D, 2D, or 3D arrays.

- A 1D array is allocated if the height and depth extent are both zero. For 1D arrays valid extent ranges are {(1, 8192), 0, 0}.
- A 2D array is allocated if only the depth extent is zero. For 2D arrays valid extent ranges are {(1, 65536), (1, 32768), 0}.
- A 3D array is allocated if all three extents are non-zero. For 3D arrays valid extent ranges are {(1, 2048), (1, 2048)}.

Note:

Due to the differing extent limits, it may be advantageous to use a degenerate array (with unused dimensions set to one) of higher dimensionality. For instance, a degenerate 2D array allows for significantly more linear storage than a 1D array.

Parameters:

```
arrayPtr - Pointer to allocated array in device memorydesc - Requested channel formatextent - Requested allocation size
```

Returns:

cudaSuccess, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc3D, cudaMallocPitch, cudaFree, cudaFreeArray, cudaMallocHost, cudaFreeHost, cudaHostAlloc, make cudaExtent

4.8.2.12 cudaError_t cudaMallocArray (struct cudaArray ** arrayPtr, const struct cudaChannelFormatDesc * desc, size_t width, size_t height)

Allocates a CUDA array according to the cudaChannelFormatDesc structure desc and returns a handle to the new CUDA array in *array.

The cudaChannelFormatDesc is defined as:

```
struct cudaChannelFormatDesc {
    int x, y, z, w;
enum cudaChannelFormatKind f;
};
```

where cudaChannelFormatKind is one of cudaChannelFormatKindSigned, cudaChannelFormatKindUnsigned, or cudaChannelFormatKindFloat.

Parameters:

```
arrayPtr - Pointer to allocated array in device memorydesc - Requested channel formatwidth - Requested array allocation width
```

Returns:

cudaSuccess, cudaErrorMemoryAllocation

height - Requested array allocation height

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc, cudaMallocPitch, cudaFree, cudaFreeArray, cudaMallocHost, cudaFreeHost, cudaMalloc3D, cudaMalloc3DArray, cudaHostAlloc

4.8.2.13 cudaError_t cudaMallocHost (void ** ptr, size_t size)

Allocates size bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as cudaMemcpy*(). Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as malloc(). Allocating excessive amounts of memory with cudaMallocHost() may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

Parameters:

```
ptr - Pointer to allocated host memorysize - Requested allocation size in bytes
```

Returns:

cudaSuccess, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc, cudaMallocPitch, cudaMallocArray, cudaMalloc3D, cudaMalloc3DArray, cudaHostAlloc, cudaFree, cudaFreeArray, cudaFreeHost, cudaHostAlloc

4.8.2.14 cudaError_t cudaMallocPitch (void ** devPtr, size_t * pitch, size_t width, size_t height)

Allocates at least widthInBytes * height bytes of linear memory on the device and returns in *devPtr a pointer to the allocated memory. The function may pad the allocation to ensure that corresponding pointers in any given row will continue to meet the alignment requirements for coalescing as the address is updated from row to row. The pitch returned in *pitch by cudaMallocPitch() is the width in bytes of the allocation. The intended usage of pitch is as a separate parameter of the allocation, used to compute addresses within the 2D array. Given the row and column of an array element of type T, the address is computed as:

```
T* pElement = (T*)((char*)BaseAddress + Row * pitch) + Column;
```

For allocations of 2D arrays, it is recommended that programmers consider performing pitch allocations using cud-aMallocPitch(). Due to pitch alignment restrictions in the hardware, this is especially true if the application will be performing 2D memory copies between different regions of device memory (whether linear memory or CUDA arrays).

Parameters:

```
    devPtr - Pointer to allocated pitched device memory
    pitch - Pitch for allocation
    width - Requested pitched allocation width
    height - Requested pitched allocation height
```

Returns:

cudaSuccess, cudaErrorMemoryAllocation

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc, cudaFree, cudaMallocArray, cudaFreeArray, cudaMallocHost, cudaFreeHost, cudaMalloc3D, cudaMalloc3DArray, cudaHostAlloc

4.8.2.15 cudaError_t cudaMemcpy (void * dst, const void * src, size_t count, enum cudaMemcpyKind kind)

Copies count bytes from the memory area pointed to by src to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. The memory areas may not overlap. Calling cudaMemcpy() with dst and src pointers that do not match the direction of the copy results in an undefined behavior.

Parameters:

```
dst - Destination memory address
src - Source memory address
count - Size in bytes to copy
kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.16 cudaError_t cudaMemcpy2D (void * dst, size_t dpitch, const void * src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies a matrix (height rows of width bytes each) from the memory area pointed to by src to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. dpitch and spitch are the widths in memory in bytes of the 2D arrays pointed to by dst and src, including any padding added to the end of each row. The memory areas may not overlap. Calling cudaMemcpy2D() with dst and src pointers that do not match the direction of the copy results in an undefined behavior. cudaMemcpy2D() returns an error if dpitch or spitch is greater than the maximum allowed.

Parameters:

```
    dst - Destination memory address
    dpitch - Pitch of destination memory
    src - Source memory address
    spitch - Pitch of source memory
    width - Width of matrix transfer (columns in bytes)
    height - Height of matrix transfer (rows)
    kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidPitchValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpy-ToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.17 cudaError_t cudaMemcpy2DArrayToArray (struct cudaArray * dst, size_t wOffsetDst, size_t hOffsetDst, const struct cudaArray * src, size_t wOffsetSrc, size_t hOffsetSrc, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies a matrix (height rows of width bytes each) from the CUDA array srcArray starting at the upper left corner (wOffsetSrc, hOffsetSrc) to the CUDA array dst starting at the upper left corner (wOffsetDst, hOffsetDst), where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDevice-ToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy.

Parameters:

```
dst - Destination memory address
wOffsetDst - Destination starting X offset
hOffsetDst - Destination starting Y offset
src - Source memory address
wOffsetSrc - Source starting X offset
hOffsetSrc - Source starting Y offset
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.18 cudaError_t cudaMemcpy2DAsync (void * dst, size_t dpitch, const void * src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies a matrix (height rows of width bytes each) from the memory area pointed to by src to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. dpitch and spitch are the widths in memory in bytes of the 2D arrays pointed to by dst and src, including any padding added to the end of each row. The memory areas may not overlap. Calling cudaMemcpy2DAsync() with dst and src pointers that do not match the direction of the copy results in an undefined behavior. cudaMemcpy2DAsync() returns an error if dpitch or spitch is greater than the maximum allowed.

cudaMemcpy2DAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
    dst - Destination memory address
    dpitch - Pitch of destination memory
    src - Source memory address
    spitch - Pitch of source memory
    width - Width of matrix transfer (columns in bytes)
    height - Height of matrix transfer (rows)
    kind - Type of transfer
    stream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidPitchValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.19 cudaError_t cudaMemcpy2DFromArray (void * dst, size_t dpitch, const struct cudaArray * src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies a matrix (height rows of width bytes each) from the CUDA array srcArray starting at the upper left corner (wOffset, hOffset) to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. dpitch is the width in memory in bytes of the 2D array pointed to by dst, including any padding added to the end of each row. cudaMemcpy2DFromArray() returns an error if dpitch is greater than the maximum allowed.

Parameters:

```
dst - Destination memory address
dpitch - Pitch of destination memory
src - Source memory address
wOffset - Source starting X offset
hOffset - Source starting Y offset
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidPitchValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.20 cudaError_t cudaMemcpy2DFromArrayAsync (void * dst, size_t dpitch, const struct cudaArray * src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies a matrix (height rows of width bytes each) from the CUDA array srcArray starting at the upper left corner (wOffset, hOffset) to the memory area pointed to by dst, where kind is one of cudaMemcpyHost-ToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. dpitch is the width in memory in bytes of the 2D array pointed to by dst, including any padding added to the end of each row. cudaMemcpy2DFromArrayAsync() returns an error if dpitch is greater than the maximum allowed.

cudaMemcpy2DFromArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
    dst - Destination memory address
    dpitch - Pitch of destination memory
    src - Source memory address
    wOffset - Source starting X offset
    hOffset - Source starting Y offset
    width - Width of matrix transfer (columns in bytes)
    height - Height of matrix transfer (rows)
    kind - Type of transfer
    stream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidPitchValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyFromSymbolAsync

4.8.2.21 cudaError_t cudaMemcpy2DToArray (struct cudaArray * dst, size_t wOffset, size_t hOffset, const void * src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies a matrix (height rows of width bytes each) from the memory area pointed to by src to the CUDA array dst starting at the upper left corner (wOffset, hOffset) where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. spitch is the width in memory in bytes of the 2D array pointed to by src, including any padding added to the end of each row. cudaMemcpy2DToArray() returns an error if spitch is greater than the maximum allowed.

Parameters:

```
dst - Destination memory address
wOffset - Destination starting X offset
hOffset - Destination starting Y offset
src - Source memory address
spitch - Pitch of source memory
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidPitchValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.22 cudaError_t cudaMemcpy2DToArrayAsync (struct cudaArray * dst, size_t wOffset, size_t hOffset, const void * src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream t stream)

Copies a matrix (height rows of width bytes each) from the memory area pointed to by src to the CUDA array dst starting at the upper left corner (wOffset, hOffset) where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. spitch is the width in memory in bytes of the 2D array pointed to by src, including any padding added to the end of each row. cudaMemcpy2DToArrayAsync() returns an error if spitch is greater than the maximum allowed.

cudaMemcpy2DToArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
dst - Destination memory address
wOffset - Destination starting X offset
hOffset - Destination starting Y offset
src - Source memory address
spitch - Pitch of source memory
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer
stream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidPitchValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.23 cudaError_t cudaMemcpy3D (const struct cudaMemcpy3DParms * p)

```
struct cudaExtent {
    size_t width;
```

```
size t height;
 size_t depth;
}:
struct cudaExtent make_cudaExtent(size_t w, size_t h, size_t d);
struct cudaPos {
 size_t x;
 size_t y;
 size_t z;
struct cudaPos make_cudaPos(size_t x, size_t y, size_t z);
struct cudaMemcpy3DParms {
  struct cudaArray *srcArray;
  struct cudaPos
                      srcPos:
  struct cudaPitchedPtr srcPtr;
 struct cudaArray *dstArray;
 struct cudaPos
                      dstPos:
 struct cudaPitchedPtr dstPtr;
 struct cudaExtent extent:
 enum cudaMemcpyKind
```

cudaMemcpy3D() copies data betwen two 3D objects. The source and destination objects may be in either host memory, device memory, or a CUDA array. The source, destination, extent, and kind of copy performed is specified by the cudaMemcpy3DParms struct which should be initialized to zero before use:

```
cudaMemcpy3DParms myParms = {0};
```

The struct passed to cudaMemcpy3D() must specify one of srcArray or srcPtr and one of dstArray or dstPtr. Passing more than one non-zero source or destination will cause cudaMemcpy3D() to return an error.

The srcPos and dstPos fields are optional offsets into the source and destination objects and are defined in units of each object's elements. The element for a host or device pointer is assumed to be **unsigned char**. For CUDA arrays, positions must be in the range [0, 2048) for any dimension.

The extent field defines the dimensions of the transferred area in elements. If a CUDA array is participating in the copy, the extent is defined in terms of that array's elements. If no CUDA array is participating in the copy then the extents are defined in elements of **unsigned char**.

The kind field defines the direction of the copy. It must be one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice.

If the source and destination are both arrays, cudaMemcpy3D() will return an error if they do not have the same element size.

The source and destination object may not overlap. If overlapping source and destination objects are specified, undefined behavior will result.

cudaMemcpy3D() returns an error if the pitch of srcPtr or dstPtr is greater than the maximum allowed. The pitch of a cudaPitchedPtr allocated with cudaMalloc3D() will always be valid.

Parameters:

p - 3D memory copy parameters

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidPitchValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc3D, cudaMalloc3DArray, cudaMemset3D, cudaMemcpy3DAsync, cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy4DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpy2DAsync, cudaMemcpy2DAsync, cudaMemcpy2DToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy5DFromArrayAsync, cudaMemcpy5DFromArrayAsync, cudaMemcpy5DFromArrayAsync, cudaMemcpy5DFromArrayAsync, cudaMemcpy5DFromSymbolAsync, make_cudaExtent, make_cudaPos

4.8.2.24 cudaError_t cudaMemcpy3DAsync (const struct cudaMemcpy3DParms * p, cudaStream_t stream)

```
struct cudaExtent {
 size_t width;
 size_t height;
 size t depth:
struct cudaExtent make_cudaExtent(size_t w, size_t h, size_t d);
struct cudaPos {
 size_t x;
 size_t y;
 size_t z;
struct cudaPos make_cudaPos(size_t x, size_t y, size_t z);
struct cudaMemcpv3DParms {
 struct cudaArray *srcArray;
 struct cudaPos
                       srcPos:
 struct cudaPitchedPtr srcPtr;
 struct cudaArray *dstArray;
 struct cudaPos
                       dstPos;
 struct cudaPitchedPtr dstPtr;
 struct cudaExtent extent;
 enum cudaMemcpyKind kind;
};
```

cudaMemcpy3DAsync() copies data betwen two 3D objects. The source and destination objects may be in either host memory, device memory, or a CUDA array. The source, destination, extent, and kind of copy performed is specified by the cudaMemcpy3DParms struct which should be initialized to zero before use:

```
cudaMemcpy3DParms myParms = {0};
```

The struct passed to cudaMemcpy3DAsync() must specify one of srcArray or srcPtr and one of dstArray or dstPtr. Passing more than one non-zero source or destination will cause cudaMemcpy3DAsync() to return an error.

The srcPos and dstPos fields are optional offsets into the source and destination objects and are defined in units of each object's elements. The element for a host or device pointer is assumed to be **unsigned char**. For CUDA arrays, positions must be in the range [0, 2048) for any dimension.

The extent field defines the dimensions of the transferred area in elements. If a CUDA array is participating in the copy, the extent is defined in terms of that array's elements. If no CUDA array is participating in the copy then the extents are defined in elements of **unsigned char**.

The kind field defines the direction of the copy. It must be one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice.

If the source and destination are both arrays, cudaMemcpy3DAsync() will return an error if they do not have the same element size.

The source and destination object may not overlap. If overlapping source and destination objects are specified, undefined behavior will result.

cudaMemcpy3DAsync() returns an error if the pitch of srcPtr or dstPtr is greater than the maximum allowed. The pitch of a cudaPitchedPtr allocated with cudaMalloc3D() will always be valid.

cudaMemcpy3DAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
p - 3D memory copy parametersstream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidPitchValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMalloc3D, cudaMalloc3DArray, cudaMemset3D, cudaMemcpy3D, cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DFromArray, cudaMemcpyToArray, cudaMemcpyToArray, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync, make_cudaExtent, make_cudaPos

4.8.2.25 cudaError_t cudaMemcpyArrayToArray (struct cudaArray * dst, size_t wOffsetDst, size_t hOffsetDst, const struct cudaArray * src, size_t wOffsetSrc, size_t hOffsetSrc, size_t count, enum cudaMemcpyKind kind)

Copies count bytes from the CUDA array src starting at the upper left corner (wOffsetSrc, hOffsetSrc) to the CUDA array dst starting at the upper left corner (wOffsetDst, hOffsetDst) where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy.

Parameters:

```
dst - Destination memory address
wOffsetDst - Destination starting X offset
hOffsetDst - Destination starting Y offset
src - Source memory address
wOffsetSrc - Source starting X offset
hOffsetSrc - Source starting Y offset
count - Size in bytes to copy
```

kind - Type of transfer

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.26 cudaError_t cudaMemcpyAsync (void * dst, const void * src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies count bytes from the memory area pointed to by src to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy. The memory areas may not overlap. Calling cudaMemcpyAsync() with dst and src pointers that do not match the direction of the copy results in an undefined behavior.

cudaMemcpyAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpy-HostToDevice or cudaMemcpyDeviceToHost and the stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

dst - Destination memory address
src - Source memory address
count - Size in bytes to copy
kind - Type of transfer
stream - Stream identifier

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.27 cudaError_t cudaMemcpyFromArray (void * dst, const struct cudaArray * src, size_t wOffset, size t hOffset, size t count, enum cudaMemcpyKind kind)

Copies count bytes from the CUDA array src starting at the upper left corner (wOffset, hOffset) to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy.

Parameters:

dst - Destination memory address
src - Source memory address
wOffset - Source starting X offset
hOffset - Source starting Y offset
count - Size in bytes to copy
kind - Type of transfer

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpy2DAsync, cudaMemcpy2DAsync, cudaMemcpyToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.28 cudaError_t cudaMemcpyFromArrayAsync (void * dst, const struct cudaArray * src, size_t wOffset, size_t hOffset, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies count bytes from the CUDA array src starting at the upper left corner (wOffset, hOffset) to the memory area pointed to by dst, where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy.

cudaMemcpyFromArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

dst - Destination memory address
 src - Source memory address
 wOffset - Source starting X offset

```
hOffset - Source starting Y offsetcount - Size in bytes to copykind - Type of transferstream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpy2DToArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.29 cudaError_t cudaMemcpyFromSymbol (void * dst, const char * symbol, size_t count, size_t offset, enum cudaMemcpyKind kind)

Copies count bytes from the memory area pointed to by offset bytes from the start of symbol symbol to the memory area pointed to by dst. The memory areas may not overlap. symbol can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. kind can be either cudaMemcpyDeviceToHost or cudaMemcpyDeviceToDevice.

Parameters:

```
    dst - Destination memory address
    symbol - Symbol source from device
    count - Size in bytes to copy
    offset - Offset from start of symbol in bytes
    kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidSymbol, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpy2DArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.30 cudaError_t cudaMemcpyFromSymbolAsync (void * dst, const char * symbol, size_t count, size_t offset, enum cudaMemcpyKind kind, cudaStream t stream)

Copies count bytes from the memory area pointed to by offset bytes from the start of symbol symbol to the memory area pointed to by dst. The memory areas may not overlap. symbol can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. kind can be either cudaMemoryDeviceToHost or cudaMemoryDeviceToDevice.

cudaMemcpyFromSymbolAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
    dst - Destination memory address
    symbol - Symbol source from device
    count - Size in bytes to copy
    offset - Offset from start of symbol in bytes
    kind - Type of transfer
    stream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidSymbol, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpy2DToArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpy5DFromArrayAsync, cudaMemcpy5DFrom

4.8.2.31 cudaError_t cudaMemcpyToArray (struct cudaArray * dst, size_t wOffset, size_t hOffset, const void * src, size_t count, enum cudaMemcpyKind kind)

Copies count bytes from the memory area pointed to by src to the CUDA array dst starting at the upper left corner (wOffset, hOffset), where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToDevice, and specifies the direction of the copy.

Parameters:

```
dst - Destination memory addresswOffset - Destination starting X offsethOffset - Destination starting Y offset
```

```
src - Source memory addresscount - Size in bytes to copykind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyToSymbol, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpy2DAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.32 cudaError_t cudaMemcpyToArrayAsync (struct cudaArray * dst, size_t wOffset, size_t hOffset, const void * src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies count bytes from the memory area pointed to by src to the CUDA array dst starting at the upper left corner (wOffset, hOffset), where kind is one of cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, or cudaMemcpyDeviceToDevice, and specifies the direction of the copy.

cudaMemcpyToArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
    dst - Destination memory address
    wOffset - Destination starting X offset
    hOffset - Destination starting Y offset
    src - Source memory address
    count - Size in bytes to copy
    kind - Type of transfer
    stream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.33 cudaError_t cudaMemcpyToSymbol (const char * symbol, const void * src, size_t count, size_t offset, enum cudaMemcpyKind kind)

Copies count bytes from the memory area pointed to by src to the memory area pointed to by offset bytes from the start of symbol symbol. The memory areas may not overlap. symbol can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. kind can be either cudaMemopyHostToDevice or cudaMemopyDeviceToDevice.

Parameters:

```
    symbol - Symbol destination on device
    src - Source memory address
    count - Size in bytes to copy
    offset - Offset from start of symbol in bytes
    kind - Type of transfer
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidSymbol, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DFromArray, cudaMemcpy2DArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpy2DToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyToSymbolAsync, cudaMemcpyFromSymbolAsync

4.8.2.34 cudaError_t cudaMemcpyToSymbolAsync (const char * symbol, const void * src, size_t count, size_t offset, enum cudaMemcpyKind kind, cudaStream_t stream)

Copies count bytes from the memory area pointed to by src to the memory area pointed to by offset bytes from the start of symbol symbol. The memory areas may not overlap. symbol can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. kind can be either cudaMemcpyHostToDevice or cudaMemcpyDeviceToDevice.

cudaMemcpyToSymbolAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input. The copy can optionally be associated to a stream by passing a non-zero stream argument. If kind is cudaMemcpyHostToDevice and stream is non-zero, the copy may overlap with operations in other streams.

IMPORTANT NOTE: Copies with kind == cudaMemcpyDeviceToDevice are asynchronous with respect to the host, but never overlap with kernel execution.

Parameters:

```
    symbol - Symbol destination on device
    src - Source memory address
    count - Size in bytes to copy
    offset - Offset from start of symbol in bytes
    kind - Type of transfer
    stream - Stream identifier
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidSymbol, cudaErrorInvalidDevicePointer, cudaErrorInvalidMemcpyDirection

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemcpy, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpy2DToArray, cudaMemcpyFromArray, cudaMemcpy2DFromArray, cudaMemcpyArrayToArray, cudaMemcpy2DArrayToArray, cudaMemcpyToSymbol, cudaMemcpyFromSymbol, cudaMemcpyAsync, cudaMemcpy2DAsync, cudaMemcpy2DToArrayAsync, cudaMemcpyFromArrayAsync, cudaMemcpy2DFromArrayAsync, cudaMemcpyFromSymbolAsync

4.8.2.35 cudaError_t cudaMemGetInfo (size_t * free, size_t * total)

Returns in *free and *total respectively, the free and total amount of memory available for allocation by the device in bytes.

Parameters:

```
free - Returned free memory in bytestotal - Returned total memory in bytes
```

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorPriorLaunchFailure, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

4.8.2.36 cudaError_t cudaMemset (void * devPtr, int value, size_t count)

Fills the first count bytes of the memory area pointed to by devPtr with the constant byte value value.

Parameters:

devPtr - Pointer to device memory

```
value - Value to set for each byte of specified memorycount - Size in bytes to set
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemset2D, cudaMemset3D

4.8.2.37 cudaError_t cudaMemset2D (void * devPtr, size_t pitch, int value, size_t width, size_t height)

Sets to the specified value value a matrix (height rows of width bytes each) pointed to by dstPtr. pitch is the width in bytes of the 2D array pointed to by dstPtr, including any padding added to the end of each row. This function performs fastest when the pitch is one that has been passed back by cudaMallocPitch().

Parameters:

```
devPtr - Pointer to 2D device memory
pitch - Pitch in bytes of 2D device memory
value - Value to set for each byte of specified memory
width - Width of matrix set (columns in bytes)
height - Height of matrix set (rows)
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemset, cudaMemset3D

4.8.2.38 cudaError_t cudaMemset3D (struct cudaPitchedPtr pitchedDevPtr, int value, struct cudaExtent extent)

Initializes each element of a 3D array to the specified value value. The object to initialize is defined by pitchedDevPtr. The pitch field of pitchedDevPtr is the width in memory in bytes of the 3D array pointed to by pitchedDevPtr, including any padding added to the end of each row. The xsize field specifies the logical width of each row in bytes, while the ysize field specifies the height of each 2D slice in rows.

The extents of the initialized region are specified as a width in bytes, a height in rows, and a depth in slices.

Extents with width greater than or equal to the xsize of pitchedDevPtr may perform significantly faster than extents narrower than the xsize. Secondarily, extents with height equal to the ysize of pitchedDevPtr will perform faster than when the height is shorter than the ysize.

This function performs fastest when the pitchedDevPtr has been allocated by cudaMalloc3D().

Parameters:

```
pitchedDevPtr - Pointer to pitched device memoryvalue - Value to set for each byte of specified memoryextent - Size parameters for where to set device memory
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaMemset, cudaMemset2D, cudaMalloc3D, make_cudaPitchedPtr, make_cudaExtent

4.8.2.39 struct cudaExtent make_cudaExtent (size_t w, size_t h, size_t d) [read]

Returns a cudaExtent based on the specified input parameters w, h, and d.

Parameters:

- w Width in bytes
- h Height in bytes
- d Depth in bytes

Returns:

cudaExtent specified by w, h, and d

See also:

make_cudaPitchedPtr, make_cudaPos

4.8.2.40 struct cudaPitchedPtr make_cudaPitchedPtr (void * d, size_t p, size_t xsz, size_t ysz) [read]

Returns a cudaPitchedPtr based on the specified input parameters d, p, xsz, and ysz.

Parameters:

- d Pointer to allocated memory
- p Pitch of allocated memory in bytes
- xsz Logical width of allocation in elements
- ysz Logical height of allocation in elements

Returns:

```
cudaPitchedPtr specified by d, p, xsz, and ysz
```

See also:

make_cudaExtent, make_cudaPos

4.8.2.41 struct cudaPos make_cudaPos (size_t *x*, **size_t** *y*, **size_t** *z*) [read]

Returns a cudaPos based on the specified input parameters x, y, and z.

Parameters:

- x X position
- y Y position
- z Z position

Returns:

cudaPos specified by x, y, and z

See also:

make_cudaExtent, make_cudaPitchedPtr

4.9 OpenGL Interoperability

Modules

• OpenGL Interoperability [DEPRECATED]

Functions

cudaError_t cudaGLSetGLDevice (int device)
 Sets the CUDA device for use with OpenGL interoperability.

cudaError_t cudaGraphicsGLRegisterBuffer (struct cudaGraphicsResource **resource, GLuint buffer, unsigned int flags)

Registers an OpenGL buffer object.

• cudaError_t cudaGraphicsGLRegisterImage (struct cudaGraphicsResource **resource, GLuint image, GLenum target, unsigned int flags)

Register an OpenGL texture or renderbuffer object.

• cudaError_t cudaWGLGetDevice (int *device, HGPUNV hGpu)

Gets the CUDA device associated with hGpu.

4.9.1 Detailed Description

This section describes the OpenGL interoperability functions of the CUDA runtime application programming interface.

4.9.2 Function Documentation

4.9.2.1 cudaError_t cudaGLSetGLDevice (int device)

Records device as the device on which the active host thread executes the device code. Records the thread as using OpenGL interoperability. If the host thread has already initialized the CUDA runtime by calling non-device management runtime functions, this call returns cudaErrorSetOnActiveProcess.

Parameters:

device - Device to use for OpenGL interoperability

Returns:

cuda Success, cuda Error Invalid Device, cuda Error Set On Active Process

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda GLR egister Buffer Object, cuda GLUn map Buffer Object Async, cuda GLUN map Buffer Object Async,

4.9.2.2 cudaError_t cudaGraphicsGLRegisterBuffer (struct cudaGraphicsResource ** resource, GLuint buffer, unsigned int flags)

Registers the buffer object specified by buffer for access by CUDA. A handle to the registered object is returned as resource. The map flags Flags specify the intended usage, as follows:

- cudaGraphicsMapFlagsNone: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA. This is the default value.
- cudaGraphicsMapFlagsReadOnly: Specifies that CUDA will not write to this resource.
- cudaGraphicsMapFlagsWriteDiscard: Specifies that CUDA will not read from this resource and will write over the entire contents of the resource, so none of the data previously stored in the resource will be preserved.

Parameters:

```
resource - Pointer to the returned object handlebuffer - name of buffer object to be registeredflags - Map flags
```

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda GLCtx Create, \quad cuda Graphics Unregister Resource, \quad cuda Graphics Map Resources, \quad cuda Graphics Resource Get Mapped Pointer$

4.9.2.3 cudaError_t cudaGraphicsGLRegisterImage (struct cudaGraphicsResource ** resource, GLuint image, GLenum target, unsigned int flags)

Registers the texture or renderbuffer object specified by image for access by CUDA. target must match the type of the object. A handle to the registered object is returned as resource. The map flags Flags specify the intended usage, as follows:

- cudaGraphicsMapFlagsNone: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA. This is the default value.
- cudaGraphicsMapFlagsReadOnly: Specifies that CUDA will not write to this resource.
- cudaGraphicsMapFlagsWriteDiscard: Specifies that CUDA will not read from this resource and will write over the entire contents of the resource, so none of the data previously stored in the resource will be preserved.

The following image classes are currently disallowed:

- Textures with borders
- · Multisampled renderbuffers

Parameters:

resource - Pointer to the returned object handle

image - name of texture or renderbuffer object to be registered

target - Identifies the type of object specified by image, and must be one of GL_TEXTURE_2D, GL_TEXTURE_RECTANGLE, GL_TEXTURE_CUBE_MAP, GL_TEXTURE_3D, GL_TEXTURE_2D_ARRAY, or GL_RENDERBUFFER.

flags - Map flags

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGLSetGLDevice cudaGraphicsUnregisterResource, cudaGraphicsMapResources, cudaGraphicsSubResourceGetMappedArray

4.9.2.4 cudaError_t cudaWGLGetDevice (int * device, HGPUNV hGpu)

Returns the CUDA device associated with a hGpu, if applicable.

Parameters:

device - Returns the device associated with hGpu, or -1 if hGpu is not a compute device.

hGpu - Handle to a GPU, as queried via WGL_NV_gpu_affinity()

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

WGL_NV_gpu_affinity, cudaGLSetGLDevice

4.10 OpenGL Interoperability [DEPRECATED]

Functions

cudaError_t cudaGLMapBufferObject (void **devPtr, GLuint bufObj)
 Maps a buffer object for access by CUDA.

• cudaError_t cudaGLMapBufferObjectAsync (void **devPtr, GLuint bufObj, cudaStream_t stream)

Maps a buffer object for access by CUDA.

• cudaError_t cudaGLRegisterBufferObject (GLuint bufObj)

Registers a buffer object for access by CUDA.

• cudaError_t cudaGLSetBufferObjectMapFlags (GLuint bufObj, unsigned int flags)

Set usage flags for mapping an OpenGL buffer.

cudaError_t cudaGLUnmapBufferObject (GLuint bufObj)

Unmaps a buffer object for access by CUDA.

• cudaError_t cudaGLUnmapBufferObjectAsync (GLuint bufObj, cudaStream_t stream)

Unmaps a buffer object for access by CUDA.

cudaError_t cudaGLUnregisterBufferObject (GLuint bufObj)

Unregisters a buffer object for access by CUDA.

4.10.1 Detailed Description

This section describes deprecated OpenGL interoperability functionality.

4.10.2 Function Documentation

4.10.2.1 cudaError_t cudaGLMapBufferObject (void ** devPtr, GLuint bufObj)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the buffer object of ID bufObj into the address space of CUDA and returns in *devPtr the base pointer of the resulting mapping. The buffer must have previously been registered by calling cudaGLRegisterBufferObject(). While a buffer is mapped by CUDA, any OpenGL operation which references the buffer will result in undefined behavior. The OpenGL context used to create the buffer, or another context from the same share group, must be bound to the current thread when this is called.

All streams in the current thread are synchronized with the current GL context.

Parameters:

devPtr - Returned device pointer to CUDA object

bufObj - Buffer object ID to map

Returns:

cudaSuccess, cudaErrorMapBufferObjectFailed

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsMapResources

4.10.2.2 cudaError_t cudaGLMapBufferObjectAsync (void ** devPtr, GLuint bufObj, cudaStream_t stream)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the buffer object of ID bufObj into the address space of CUDA and returns in *devPtr the base pointer of the resulting mapping. The buffer must have previously been registered by calling cudaGLRegisterBufferObject(). While a buffer is mapped by CUDA, any OpenGL operation which references the buffer will result in undefined behavior. The OpenGL context used to create the buffer, or another context from the same share group, must be bound to the current thread when this is called.

Stream /p stream is synchronized with the current GL context.

Parameters:

devPtr - Returned device pointer to CUDA object

 $\mathit{bufObj}\,$ - Buffer object ID to map

stream - Stream to synchronize

Returns:

cudaSuccess, cudaErrorMapBufferObjectFailed

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Map Resources

4.10.2.3 cudaError_t cudaGLRegisterBufferObject (GLuint bufObj)

Deprecated

This function is deprecated as of Cuda 3.0.

Registers the buffer object of ID bufObj for access by CUDA. This function must be called before CUDA can map the buffer object. The OpenGL context used to create the buffer, or another context from the same share group, must be bound to the current thread when this is called.

Parameters:

bufObj - Buffer object ID to register

Returns:

cudaSuccess, cudaErrorInitializationError

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsGLRegisterBuffer

4.10.2.4 cudaError t cudaGLSetBufferObjectMapFlags (GLuint bufObj, unsigned int flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Set flags for mapping the OpenGL buffer buf0bj

Changes to flags will take effect the next time bufObj is mapped. The flags argument may be any of the following:

- cudaGLMapFlagsNone: Specifies no hints about how this buffer will be used. It is therefore assumed that this buffer will be read from and written to by CUDA kernels. This is the default value.
- cudaGLMapFlagsReadOnly: Specifies that CUDA kernels which access this buffer will not write to the buffer.
- cudaGLMapFlagsWriteDiscard: Specifies that CUDA kernels which access this buffer will not read from the buffer and will write over the entire contents of the buffer, so none of the data previously stored in the buffer will be preserved.

If bufObj has not been registered for use with CUDA, then cudaErrorInvalidResourceHandle is returned. If bufObj is presently mapped for access by CUDA, then cudaErrorUnknown is returned.

Parameters:

bufObj - Registered buffer object to set flags for

flags - Parameters for buffer mapping

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Resource Set Map Flags

4.10.2.5 cudaError_t cudaGLUnmapBufferObject (GLuint bufObj)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the buffer object of ID bufObj for access by CUDA. When a buffer is unmapped, the base address returned by cudaGLMapBufferObject() is invalid and subsequent references to the address result in undefined behavior. The OpenGL context used to create the buffer, or another context from the same share group, must be bound to the current thread when this is called.

All streams in the current thread are synchronized with the current GL context.

Parameters:

bufObj - Buffer object to unmap

Returns:

cudaSuccess, cudaErrorInvalidDevicePointer, cudaErrorUnmapBufferObjectFailed

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsUnmapResources

4.10.2.6 cudaError_t cudaGLUnmapBufferObjectAsync (GLuint bufObj, cudaStream_t stream)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the buffer object of ID <code>bufObj</code> for access by CUDA. When a buffer is unmapped, the base address returned by <code>cudaGLMapBufferObject()</code> is invalid and subsequent references to the address result in undefined behavior. The OpenGL context used to create the buffer, or another context from the same share group, must be bound to the current thread when this is called.

Stream /p stream is synchronized with the current GL context.

Parameters:

bufObj - Buffer object to unmapstream - Stream to synchronize

Returns:

cudaSuccess, cudaErrorInvalidDevicePointer, cudaErrorUnmapBufferObjectFailed

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsUnmapResources

4.10.2.7 cudaError_t cudaGLUnregisterBufferObject (GLuint bufObj)

Deprecated

This function is deprecated as of Cuda 3.0.

Unregisters the buffer object of ID bufObj for access by CUDA and releases any CUDA resources associated with the buffer. Once a buffer is unregistered, it may no longer be mapped by CUDA. The GL context used to create the buffer, or another context from the same share group, must be bound to the current thread when this is called.

Parameters:

bufObj - Buffer object to unregister

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Unregister Resource

4.11 Direct3D 9 Interoperability

Modules

• Direct3D 9 Interoperability [DEPRECATED]

Functions

- cudaError_t cudaD3D9GetDevice (int *device, const char *pszAdapterName)
 Gets the device number for an adapter.
- cudaError_t cudaD3D9SetDirect3DDevice (IDirect3DDevice9 *pD3D9Device)

Sets the Direct3D device to use for interoperability in this thread.

 cudaError_t cudaGraphicsD3D9RegisterResource (struct cudaGraphicsResource **resource, IDirect3DResource9 *pD3DResource, unsigned int flags)

Register a Direct3D 9 resource for access by CUDA.

4.11.1 Detailed Description

This section describes the Direct3D 9 interoperability functions of the CUDA runtime application programming interface.

4.11.2 Function Documentation

4.11.2.1 cudaError_t cudaD3D9GetDevice (int * device, const char * pszAdapterName)

Returns in *device the CUDA-compatible device corresponding to the adapter name pszAdapterName obtained from EnumDisplayDevices or IDirect3D9::GetAdapterIdentifier(). If no device on the adapter with name pszAdapterName is CUDA-compatible then the call will fail.

Parameters:

```
device - Returns the device corresponding to pszAdapterNamepszAdapterName - D3D9 adapter to get device for
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D9SetDirect3DDevice, cudaGraphicsD3D9RegisterResource,

4.11.2.2 cudaError_t cudaD3D9SetDirect3DDevice (IDirect3DDevice9 * pD3D9Device)

Records pD3D9Device as the Direct3D device to use for Direct3D interoperability on this host thread. In order to use Direct3D interoperability, this call must be made before any non-device management CUDA runtime calls on this thread. In that case, this call will return cudaErrorSetOnActiveProcess.

Successful context creation on pD3D9Device will increase the internal reference count on pD3D9Device. This reference count will be decremented upon destruction of this context through cudaThreadExit().

Parameters:

pD3D9Device - Direct3D device to use for this thread

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorInvalidValue, cudaErrorSetOnActiveProcess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D9GetDevice, cudaGraphicsD3D9RegisterResource,

4.11.2.3 cudaError_t cudaGraphicsD3D9RegisterResource (struct cudaGraphicsResource ** resource, IDirect3DResource9 * pD3DResource, unsigned int flags)

Registers the Direct3D 9 resource pD3DResource for access by CUDA.

If this call is successful then the application will be able to map and unmap this resource until it is unregistered through cudaGraphicsUnregisterResource(). Also on success, this call will increase the internal reference count on pD3DResource. This reference count will be decremented when this resource is unregistered through cudaGraphicsUnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pD3DResource must be one of the following.

- IDirect3DVertexBuffer9: may be accessed through a device pointer
- IDirect3DIndexBuffer9: may be accessed through a device pointer
- IDirect3DSurface9: may be accessed through an array. Only stand-alone objects of type IDirect3DSurface9 may be explicitly shared. In particular, individual mipmap levels and faces of cube maps may not be registered directly. To access individual surfaces associated with a texture, one must register the base texture object.
- IDirect3DBaseTexture9: individual surfaces on this texture may be accessed through an array.

The flags argument may be used to specify additional parameters at register time. The only valid value for this parameter is

• cudaGraphicsRegisterFlagsNone

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized using cudaD3D9SetDirect3DDevice then cudaErrorInvalidDevice is returned. If pD3DResource is of incorrect type or is already registered, then cudaErrorInvalidResourceHandle is returned. If pD3DResource cannot be registered, then cudaErrorUnknown is returned.

Parameters:

```
resource - Pointer to returned resource handlepD3DResource - Direct3D resource to registerflags - Parameters for resource registration
```

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D9SetDirect3DDevice cudaGraphicsUnregisterResource, cudaGraphicsMapResources, cudaGraphicsSubResourceGetMappedArray, cudaGraphicsResourceGetMappedPointer

4.12 Direct3D 9 Interoperability [DEPRECATED]

Functions

• cudaError_t cudaD3D9GetDirect3DDevice (IDirect3DDevice9 **ppD3D9Device)

Gets the Direct3D device against which the current CUDA context was created.

• cudaError_t cudaD3D9MapResources (int count, IDirect3DResource9 **ppResources)

Map Direct3D resources for access by CUDA.

• cudaError_t cudaD3D9RegisterResource (IDirect3DResource9 *pResource, unsigned int flags)

Registers a Direct3D resource for access by CUDA.

• cudaError_t cudaD3D9ResourceGetMappedArray (cudaArray **ppArray, IDirect3DResource9 *pResource, unsigned int face, unsigned int level)

Get an array through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

• cudaError_t cudaD3D9ResourceGetMappedPitch (size_t *pPitch, size_t *pPitchSlice, IDirect3DResource9 *pResource, unsigned int face, unsigned int level)

Get the pitch of a subresource of a Direct3D resource which has been mapped for access by CUDA.

• cudaError_t cudaD3D9ResourceGetMappedPointer (void **pPointer, IDirect3DResource9 *pResource, unsigned int face, unsigned int level)

Get a pointer through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

• cudaError_t cudaD3D9ResourceGetMappedSize (size_t *pSize, IDirect3DResource9 *pResource, unsigned int face, unsigned int level)

Get the size of a subresource of a Direct3D resource which has been mapped for access by CUDA.

• cudaError_t cudaD3D9ResourceGetSurfaceDimensions (size_t *pWidth, size_t *pHeight, size_t *pDepth, IDirect3DResource9 *pResource, unsigned int face, unsigned int level)

Get the dimensions of a registered Direct3D surface.

• cudaError_t cudaD3D9ResourceSetMapFlags (IDirect3DResource9 *pResource, unsigned int flags)

Set usage flags for mapping a Direct3D resource.

cudaError_t cudaD3D9UnmapResources (int count, IDirect3DResource9 **ppResources)

Unmap Direct3D resources for access by CUDA.

• cudaError_t cudaD3D9UnregisterResource (IDirect3DResource9 *pResource)

Unregisters a Direct3D resource for access by CUDA.

4.12.1 Detailed Description

This section describes deprecated Direct3D 9 interoperability functions.

4.12.2 Function Documentation

4.12.2.1 cudaError_t cudaD3D9GetDirect3DDevice (IDirect3DDevice9 ** ppD3D9Device)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *ppD3D9Device the Direct3D device against which this CUDA context was created in cu-daD3D9SetDirect3DDevice().

Parameters:

ppD3D9Device - Returns the Direct3D device for this thread

Returns:

cudaSuccess, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D9SetDirect3DDevice

4.12.2.2 cudaError_t cudaD3D9MapResources (int count, IDirect3DResource9 ** ppResources)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the count Direct3D resources in ppResources for access by CUDA.

The resources in ppResources may be accessed in CUDA kernels until they are unmapped. Direct3D should not access any resources while they are mapped by CUDA. If an application does so, the results are undefined.

This function provides the synchronization guarantee that any Direct3D calls issued before cudaD3D9MapResources() will complete before any CUDA kernels issued after cudaD3D9MapResources() begin.

If any of ppResources have not been registered for use with CUDA or if ppResources contains any duplicate entries then cudaErrorInvalidResourceHandle is returned. If any of ppResources are presently mapped for access by CUDA then cudaErrorUnknown is returned.

Parameters:

```
count - Number of resources to map for CUDAppResources - Resources to map for CUDA
```

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsMapResources

4.12.2.3 cudaError_t cudaD3D9RegisterResource (IDirect3DResource9 * pResource, unsigned int flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Registers the Direct3D resource pResource for access by CUDA.

If this call is successful, then the application will be able to map and unmap this resource until it is unregistered through cudaD3D9UnregisterResource(). Also on success, this call will increase the internal reference count on pResource. This reference count will be decremented when this resource is unregistered through cudaD3D9UnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pResource must be one of the following.

- IDirect3DVertexBuffer9: No notes.
- IDirect3DIndexBuffer9: No notes.
- IDirect3DSurface9: Only stand-alone objects of type IDirect3DSurface9 may be explicitly shared. In particular, individual mipmap levels and faces of cube maps may not be registered directly. To access individual surfaces associated with a texture, one must register the base texture object.
- IDirect3DBaseTexture9: When a texture is registered, all surfaces associated with all mipmap levels of all faces
 of the texture will be accessible to CUDA.

The flags argument specifies the mechanism through which CUDA will access the Direct3D resource. The following value is allowed:

• cudaD3D9RegisterFlagsNone: Specifies that CUDA will access this resource through a void*. The pointer, size, and pitch for each subresource of this resource may be queried through cudaD3D9ResourceGetMappedPointer(), cudaD3D9ResourceGetMappedSize(), and cudaD3D9ResourceGetMappedPitch() respectively. This option is valid for all resource types.

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations:

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Any resources allocated in D3DPOOL_SYSTEMMEM or D3DPOOL_MANAGED may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized on this context, then cudaErrorInvalidDevice is returned. If pResource is of incorrect type (e.g., is a non-stand-alone IDirect3DSurface9) or is already registered, then cudaErrorInvalidResourceHandle is returned. If pResource cannot be registered then cudaErrorUnknown is returned.

Parameters:

pResource - Resource to register

flags - Parameters for resource registration

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsD3D9RegisterResource

4.12.2.4 cudaError_t cudaD3D9ResourceGetMappedArray (cudaArray ** ppArray, IDirect3DResource9 * pResource, unsigned int face, unsigned int level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pArray an array through which the subresource of the mapped Direct3D resource pResource, which corresponds to face and level may be accessed. The value set in pArray may change every time that pResource is mapped.

If pResource is not registered then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D9RegisterFlagsArray, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped, then cudaErrorUnknown is returned.

For usage requirements of face and level parameters, see cudaD3D9ResourceGetMappedPointer().

Parameters:

```
    ppArray - Returned array corresponding to subresource
    pResource - Mapped resource to access
    face - Face of resource to access
    level - Level of resource to access
```

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics SubResource Get Mapped Array

4.12.2.5 cudaError_t cudaD3D9ResourceGetMappedPitch (size_t * pPitch, size_t * pPitchSlice, IDirect3DResource9 * pResource, unsigned int face, unsigned int level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pPitch and *pPitchSlice the pitch and Z-slice pitch of the subresource of the mapped Direct3D resource pResource, which corresponds to face and level. The values set in pPitch and pPitchSlice may change every time that pResource is mapped.

The pitch and Z-slice pitch values may be used to compute the location of a sample on a surface as follows.

For a 2D surface, the byte offset of the sample at position **x**, **y** from the base pointer of the surface is:

```
y * pitch + (bytes per pixel) * x
```

For a 3D surface, the byte offset of the sample at position x, y, z from the base pointer of the surface is:

```
z* slicePitch + y * pitch + (bytes per pixel) * x
```

Both parameters pPitch and pPitchSlice are optional and may be set to NULL.

If pResource is not of type IDirect3DBaseTexture9 or one of its sub-types or if pResource has not been registered for use with CUDA, then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D9RegisterFlagsNone, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped for access by CUDA then cudaErrorUnknown is returned.

For usage requirements of face and level parameters, see cudaD3D9ResourceGetMappedPointer().

Parameters:

```
    pPitch - Returned pitch of subresource
    pPitchSlice - Returned Z-slice pitch of subresource
    pResource - Mapped resource to access
    face - Face of resource to access
    level - Level of resource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsResourceGetMappedPointer

4.12.2.6 cudaError_t cudaD3D9ResourceGetMappedPointer (void ** pPointer, IDirect3DResource9 * pResource, unsigned int face, unsigned int level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pPointer the base pointer of the subresource of the mapped Direct3D resource pResource, which corresponds to face and level. The value set in pPointer may change every time that pResource is mapped.

If pResource is not registered, then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D9RegisterFlagsNone, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped, then cudaErrorUnknown is returned.

If pResource is of type IDirect3DCubeTexture9, then face must one of the values enumerated by type D3DCUBEMAP_FACES. For all other types, face must be 0. If face is invalid, then cudaErrorInvalidValue is returned.

If pResource is of type IDirect3DBaseTexture9, then level must correspond to a valid mipmap level. Only mipmap level 0 is supported for now. For all other types level must be 0. If level is invalid, then cudaErrorInvalidValue is returned.

Parameters:

```
    pPointer - Returned pointer corresponding to subresource
    pResource - Mapped resource to access
    face - Face of resource to access
    level - Level of resource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Resource Get Mapped Pointer

4.12.2.7 cudaError_t cudaD3D9ResourceGetMappedSize (size_t * pSize, IDirect3DResource9 * pResource, unsigned int face, unsigned int level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pSize the size of the subresource of the mapped Direct3D resource pResource, which corresponds to face and level. The value set in pSize may change every time that pResource is mapped.

If pResource has not been registered for use with CUDA then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D9RegisterFlagsNone, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped for access by CUDA then cudaErrorUnknown is returned.

For usage requirements of face and level parameters, see cudaD3D9ResourceGetMappedPointer().

Parameters:

```
    pSize - Returned size of subresource
    pResource - Mapped resource to access
    face - Face of resource to access
    level - Level of resource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsResourceGetMappedPointer

4.12.2.8 cudaError_t cudaD3D9ResourceGetSurfaceDimensions (size_t * pWidth, size_t * pHeight, size_t * pDepth, IDirect3DResource9 * pResource, unsigned int face, unsigned int level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pWidth, *pHeight, and *pDepth the dimensions of the subresource of the mapped Direct3D resource pResource which corresponds to face and level.

Because anti-aliased surfaces may have multiple samples per pixel, it is possible that the dimensions of a resource will be an integer factor larger than the dimensions reported by the Direct3D runtime.

The parameters pWidth, pHeight, and pDepth are optional. For 2D surfaces, the value returned in *pDepth will be 0.

If pResource is not of type IDirect3DBaseTexture9 or IDirect3DSurface9 or if pResource has not been registered for use with CUDA, then cudaErrorInvalidResourceHandle is returned.

For usage requirements of face and level parameters, see cudaD3D9ResourceGetMappedPointer.

Parameters:

```
    pWidth - Returned width of surface
    pHeight - Returned height of surface
    pDepth - Returned depth of surface
    pResource - Registered resource to access
    face - Face of resource to access
    level - Level of resource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics SubResource Get Mapped Array

$\textbf{4.12.2.9} \quad \textbf{cudaError_t cudaD3D9ResourceSetMapFlags (IDirect3DResource9}*pResource, \ \textbf{unsigned int} \textit{flags})$

Deprecated

This function is deprecated as of Cuda 3.0.

Set flags for mapping the Direct3D resource pResource.

Changes to flags will take effect the next time presource is mapped. The flags argument may be any of the following:

• cudaD3D9MapFlagsNone: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default value.

- cudaD3D9MapFlagsReadOnly: Specifies that CUDA kernels which access this resource will not write to this
 resource.
- cudaD3D9MapFlagsWriteDiscard: Specifies that CUDA kernels which access this resource will not read from
 this resource and will write over the entire contents of the resource, so none of the data previously stored in the
 resource will be preserved.

If pResource has not been registered for use with CUDA, then cudaErrorInvalidResourceHandle is returned. If pResource is presently mapped for access by CUDA, then cudaErrorUnknown is returned.

Parameters:

```
pResource - Registered resource to set flags for flags - Parameters for resource mapping
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaInteropResourceSetMapFlags

4.12.2.10 cudaError_t cudaD3D9UnmapResources (int count, IDirect3DResource9 ** ppResources)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the count Direct3D resources in ppResources.

This function provides the synchronization guarantee that any CUDA kernels issued before cudaD3D9UnmapResources() will complete before any Direct3D calls issued after cudaD3D9UnmapResources() begin.

If any of ppResources have not been registered for use with CUDA or if ppResources contains any duplicate entries, then cudaErrorInvalidResourceHandle is returned. If any of ppResources are not presently mapped for access by CUDA then cudaErrorUnknown is returned.

Parameters:

```
count - Number of resources to unmap for CUDAppResources - Resources to unmap for CUDA
```

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsUnmapResources

4.12.2.11 cudaError_t cudaD3D9UnregisterResource (IDirect3DResource9 * pResource)

Deprecated

This function is deprecated as of Cuda 3.0.

 $Unregisters \ the \ Direct 3D \ resource \ p{\tt Resource} \ so \ it \ is \ not \ accessible \ by \ CUDA \ unless \ registered \ again.$

If pResource is not registered, then cudaErrorInvalidResourceHandle is returned.

Parameters:

pResource - Resource to unregister

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Unregister Resource

4.13 Direct3D 10 Interoperability

Modules

• Direct3D 10 Interoperability [DEPRECATED]

Functions

- cudaError_t cudaD3D10GetDevice (int *device, IDXGIAdapter *pAdapter)

 Gets the device number for an adapter.
- cudaError_t cudaD3D10SetDirect3DDevice (ID3D10Device *pD3D10Device)

Sets the Direct3D 10 device to use for interoperability in this thread.

• cudaError_t cudaGraphicsD3D10RegisterResource (struct cudaGraphicsResource **resource, ID3D10Resource *pD3DResource, unsigned int flags)

Register a Direct3D 10 resource for access by CUDA.

4.13.1 Detailed Description

This section describes the Direct3D 10 interoperability functions of the CUDA runtime application programming interface.

4.13.2 Function Documentation

4.13.2.1 cudaError_t cudaD3D10GetDevice (int * device, IDXGIAdapter * pAdapter)

Returns in *device the CUDA-compatible device corresponding to the adapter pAdapter obtained from IDXGI-Factory::EnumAdapters. This call will succeed only if a device on adapter pAdapter is Cuda-compatible.

Parameters:

```
device - Returns the device corresponding to pAdapterpAdapter - D3D10 adapter to get device for
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda D3D10 Set Direct 3DD evice, \ cuda Graphics D3D10 Register Resource,$

4.13.2.2 cudaError_t cudaD3D10SetDirect3DDevice (ID3D10Device * pD3D10Device)

Records pD3D10Device as the Direct3D 10 device to use for Direct3D 10 interoperability on this host thread. In order to use Direct3D 10 interoperability, this call must be made before any non-device management CUDA runtime calls on this thread. In that case, this call will return cudaErrorSetOnActiveProcess.

Successful context creation on pD3D10Device will increase the internal reference count on pD3D10Device. This reference count will be decremented upon destruction of this context through cudaThreadExit().

Parameters:

pD3D10Device - Direct3D device to use for interoperability

Returns:

cuda Success, cuda Error Initialization Error, cuda Error Invalid Value, cuda Error Set On Active Process

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D10GetDevice, cudaGraphicsD3D10RegisterResource,

4.13.2.3 cudaError_t cudaGraphicsD3D10RegisterResource (struct cudaGraphicsResource ** resource, ID3D10Resource ** pD3DResource, unsigned int flags)

Registers the Direct3D 10 resource pD3DResource for access by CUDA.

If this call is successful, then the application will be able to map and unmap this resource until it is unregistered through cudaGraphicsUnregisterResource(). Also on success, this call will increase the internal reference count on pD3DResource. This reference count will be decremented when this resource is unregistered through cudaGraphicsUnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pD3DResource must be one of the following.

- ID3D10Buffer: may be accessed via a device pointer
- ID3D10Texture1D: individual subresources of the texture may be accessed via arrays
- ID3D10Texture2D: individual subresources of the texture may be accessed via arrays
- ID3D10Texture3D: individual subresources of the texture may be accessed via arrays

The flags argument may be used to specify additional parameters at register time. The only valid value for this parameter is

• cudaGraphicsRegisterFlagsNone

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

The primary rendertarget may not be registered with CUDA.

- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized using cudaD3D10SetDirect3DDevice then cudaErrorInvalidDevice is returned. If pD3DResource is of incorrect type or is already registered, then cudaErrorInvalidResourceHandle is returned. If pD3DResource cannot be registered, then cudaErrorUnknown is returned.

Parameters:

```
resource - Pointer to returned resource handlepD3DResource - Direct3D resource to registerflags - Parameters for resource registration
```

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D10SetDirect3DDevice cudaGraphicsUnregisterResource, cudaGraphicsMapResources, cudaGraphicsSubResourceGetMappedArray, cudaGraphicsResourceGetMappedPointer

4.14 Direct3D 10 Interoperability [DEPRECATED]

Functions

cudaError_t cudaD3D10MapResources (int count, ID3D10Resource **ppResources)
 Map Direct3D Resources for access by CUDA.

- cudaError_t cudaD3D10RegisterResource (ID3D10Resource *pResource, unsigned int flags)

 Register a Direct3D 10 resource for access by CUDA.
- cudaError_t cudaD3D10ResourceGetMappedArray (cudaArray **ppArray, ID3D10Resource *pResource, unsigned int subResource)

Get an array through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

• cudaError_t cudaD3D10ResourceGetMappedPitch (size_t *pPitch, size_t *pPitchSlice, ID3D10Resource *pResource, unsigned int subResource)

Get the pitch of a subresource of a Direct3D resource which has been mapped for access by CUDA.

cudaError_t cudaD3D10ResourceGetMappedPointer (void **pPointer, ID3D10Resource *pResource, unsigned int subResource)

Get a pointer through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

cudaError_t cudaD3D10ResourceGetMappedSize (size_t *pSize, ID3D10Resource *pResource, unsigned int subResource)

Get the size of a subresource of a Direct3D resource which has been mapped for access by CUDA.

• cudaError_t cudaD3D10ResourceGetSurfaceDimensions (size_t *pWidth, size_t *pHeight, size_t *pDepth, ID3D10Resource *pResource, unsigned int subResource)

Get the dimensions of a registered Direct3D surface.

- cudaError_t cudaD3D10ResourceSetMapFlags (ID3D10Resource *pResource, unsigned int flags)

 Set usage flags for mapping a Direct3D resource.
- cudaError_t cudaD3D10UnmapResources (int count, ID3D10Resource **ppResources)

 *Unmaps Direct3D resources.
- cudaError_t cudaD3D10UnregisterResource (ID3D10Resource *pResource)

 *Unregisters a Direct3D resource.

4.14.1 Detailed Description

This section describes deprecated Direct3D 10 interoperability functions.

4.14.2 Function Documentation

4.14.2.1 cudaError_t cudaD3D10MapResources (int count, ID3D10Resource ** ppResources)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the count Direct3D resources in ppResources for access by CUDA.

The resources in ppResources may be accessed in CUDA kernels until they are unmapped. Direct3D should not access any resources while they are mapped by CUDA. If an application does so, the results are undefined.

This function provides the synchronization guarantee that any Direct3D calls issued before cudaD3D10MapResources() will complete before any CUDA kernels issued after cudaD3D10MapResources() begin.

If any of ppResources have not been registered for use with CUDA or if ppResources contains any duplicate entries then cudaErrorInvalidResourceHandle is returned. If any of ppResources are presently mapped for access by CUDA then cudaErrorUnknown is returned.

Parameters:

```
count - Number of resources to map for CUDAppResources - Resources to map for CUDA
```

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsMapResources

4.14.2.2 cudaError t cudaD3D10RegisterResource (ID3D10Resource * pResource, unsigned int flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Registers the Direct3D resource pResource for access by CUDA.

If this call is successful, then the application will be able to map and unmap this resource until it is unregistered through cudaD3D10UnregisterResource(). Also on success, this call will increase the internal reference count on pResource. This reference count will be decremented when this resource is unregistered through cudaD3D10UnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pResource must be one of the following:

- ID3D10Buffer: Cannot be used with flags set to cudaD3D10RegisterFlagsArray.
- ID3D10Texture1D: No restrictions.
- ID3D10Texture2D: No restrictions.
- ID3D10Texture3D: No restrictions.

The flags argument specifies the mechanism through which CUDA will access the Direct3D resource. The following values are allowed.

• cudaD3D10RegisterFlagsNone: Specifies that CUDA will access this resource through a void*. The pointer, size, and pitch for each subresource of this resource may be queried through cudaD3D10ResourceGetMappedPointer(), cudaD3D10ResourceGetMappedSize(), and cudaD3D10ResourceGetMappedPitch() respectively. This option is valid for all resource types.

• cudaD3D10RegisterFlagsArray: Specifies that CUDA will access this resource through a CUarray queried on a sub-resource basis through cuD3D10ResourceGetMappedArray(). This option is only valid for resources of type ID3D10Texture1D, ID3D10Texture2D, and ID3D10Texture3D.

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized on this context then cudaErrorInvalidDevice is returned. If pResource is of incorrect type or is already registered then cudaErrorInvalidResourceHandle is returned. If pResource cannot be registered then cudaErrorUnknown is returned.

Parameters:

```
pResource - Resource to registerflags - Parameters for resource registration
```

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsD3D10RegisterResource

4.14.2.3 cudaError_t cudaD3D10ResourceGetMappedArray (cudaArray ** ppArray, ID3D10Resource * pResource, unsigned int subResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *ppArray an array through which the subresource of the mapped Direct3D resource pResource which corresponds to subResource may be accessed. The value set in ppArray may change every time that pResource is mapped.

If pResource is not registered, then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D10RegisterFlagsArray, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped then cudaErrorUnknown is returned.

For usage requirements of the subResource parameter, see cudaD3D10ResourceGetMappedPointer().

Parameters:

```
ppArray - Returned array corresponding to subresourcepResource - Mapped resource to accesssubResource - Subresource of pResource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsSubResourceGetMappedArray

4.14.2.4 cudaError_t cudaD3D10ResourceGetMappedPitch (size_t * pPitch, size_t * pPitchSlice, ID3D10Resource * pResource, unsigned int subResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pPitch and *pPitchSlice the pitch and Z-slice pitch of the subresource of the mapped Direct3D resource pResource, which corresponds to subResource. The values set in pPitch and pPitchSlice may change every time that pResource is mapped.

The pitch and Z-slice pitch values may be used to compute the location of a sample on a surface as follows.

For a 2D surface, the byte offset of the sample at position x, y from the base pointer of the surface is:

```
y * pitch + (bytes per pixel) * x
```

For a 3D surface, the byte offset of the sample at position x, y, z from the base pointer of the surface is:

```
z* slicePitch + y * pitch + (bytes per pixel) * x
```

Both parameters pPitch and pPitchSlice are optional and may be set to NULL.

If pResource is not of type ID3D10Texture1D, ID3D10Texture2D, or ID3D10Texture3D, or if pResource has not been registered for use with CUDA, then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D10RegisterFlagsNone, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped for access by CUDA then cudaErrorUnknown is returned.

For usage requirements of the subResource parameter see cudaD3D10ResourceGetMappedPointer().

Parameters:

```
    pPitch - Returned pitch of subresource
    pPitchSlice - Returned Z-slice pitch of subresource
    pResource - Mapped resource to access
    subResource - Subresource of pResource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics SubResource Get Mapped Array

4.14.2.5 cudaError_t cudaD3D10ResourceGetMappedPointer (void ** pPointer, ID3D10Resource * pResource, unsigned int subResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pPointer the base pointer of the subresource of the mapped Direct3D resource pResource which corresponds to subResource. The value set in pPointer may change every time that pResource is mapped.

If pResource is not registered, then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags cudaD3D9RegisterFlagsNone, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped then cudaErrorUnknown is returned.

If pResource is of type ID3D10Buffer then subResource must be 0. If pResource is of any other type, then the value of subResource must come from the subresource calculation in D3D10CalcSubResource().

Parameters:

```
pPointer - Returned pointer corresponding to subresourcepResource - Mapped resource to accesssubResource - Subresource of pResource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Resource Get Mapped Pointer

4.14.2.6 cudaError_t cudaD3D10ResourceGetMappedSize (size_t * pSize, ID3D10Resource * pResource, unsigned int subResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pSize the size of the subresource of the mapped Direct3D resource pResource which corresponds to subResource. The value set in pSize may change every time that pResource is mapped.

If pResource has not been registered for use with CUDA then cudaErrorInvalidHandle is returned. If pResource was not registered with usage flags cudaD3D10RegisterFlagsNone, then cudaErrorInvalidResourceHandle is returned. If pResource is not mapped for access by CUDA then cudaErrorUnknown is returned.

For usage requirements of the subResource parameter see cudaD3D10ResourceGetMappedPointer().

Parameters:

```
pSize - Returned size of subresourcepResource - Mapped resource to accesssubResource - Subresource of pResource to access
```

Returns:

cuda Success, cuda Error Invalid Value, cuda Error Invalid Resource Handle, cuda Error Unknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsResourceGetMappedPointer

4.14.2.7 cudaError_t cudaD3D10ResourceGetSurfaceDimensions (size_t * pWidth, size_t * pHeight, size_t * pDepth, ID3D10Resource * pResource, unsigned int subResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pWidth, *pHeight, and *pDepth the dimensions of the subresource of the mapped Direct3D resource pResource which corresponds to subResource.

Because anti-aliased surfaces may have multiple samples per pixel, it is possible that the dimensions of a resource will be an integer factor larger than the dimensions reported by the Direct3D runtime.

The parameters pWidth, pHeight, and pDepth are optional. For 2D surfaces, the value returned in *pDepth will be 0.

If pResource is not of type ID3D10Texture1D, ID3D10Texture2D, or ID3D10Texture3D, or if pResource has not been registered for use with CUDA, then cudaErrorInvalidHandle is returned.

For usage requirements of subResource parameters see cudaD3D10ResourceGetMappedPointer().

Parameters:

```
    pWidth - Returned width of surface
    pHeight - Returned height of surface
    pDepth - Returned depth of surface
    pResource - Registered resource to access
    subResource - Subresource of pResource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsSubResourceGetMappedArray

4.14.2.8 cudaError_t cudaD3D10ResourceSetMapFlags (ID3D10Resource * pResource, unsigned int flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Set usage flags for mapping the Direct3D resource pResource.

Changes to flags will take effect the next time presource is mapped. The flags argument may be any of the following:

- cudaD3D10MapFlagsNone: Specifies no hints about how this resource will be used. It is therefore assumed that
 this resource will be read from and written to by CUDA kernels. This is the default value.
- cudaD3D10MapFlagsReadOnly: Specifies that CUDA kernels which access this resource will not write to this
 resource.
- cudaD3D10MapFlagsWriteDiscard: Specifies that CUDA kernels which access this resource will not read from
 this resource and will write over the entire contents of the resource, so none of the data previously stored in the
 resource will be preserved.

If pResource has not been registered for use with CUDA then cudaErrorInvalidHandle is returned. If pResource is presently mapped for access by CUDA then cudaErrorUnknown is returned.

Parameters:

```
pResource - Registered resource to set flags for flags - Parameters for resource mapping
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsResourceSetMapFlags

4.14.2.9 cudaError_t cudaD3D10UnmapResources (int count, ID3D10Resource ** ppResources)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the count Direct3D resource in ppResources.

This function provides the synchronization guarantee that any CUDA kernels issued before cudaD3D10UnmapResources() will complete before any Direct3D calls issued after cudaD3D10UnmapResources() begin.

If any of ppResources have not been registered for use with CUDA or if ppResources contains any duplicate entries, then cudaErrorInvalidResourceHandle is returned. If any of ppResources are not presently mapped for access by CUDA then cudaErrorUnknown is returned.

Parameters:

count - Number of resources to unmap for CUDAppResources - Resources to unmap for CUDA

Returns:

cuda Success, cuda Error Invalid Resource Handle, cuda Error Unknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsUnmapResources

4.14.2.10 cudaError_t cudaD3D10UnregisterResource (ID3D10Resource * pResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Unregisters the Direct3D resource resource so it is not accessible by CUDA unless registered again.

If pResource is not registered, then cudaErrorInvalidResourceHandle is returned.

Parameters:

pResource - Resource to unregister

Returns:

cuda Success, cuda Error Invalid Resource Handle, cuda Error Unknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Unregister Resource

4.15 Direct3D 11 Interoperability

Functions

cudaError_t cudaD3D11GetDevice (int *device, IDXGIAdapter *pAdapter)
 Gets the device number for an adapter.

- cudaError_t cudaD3D11SetDirect3DDevice (ID3D11Device *pD3D11Device)

 Sets the Direct3D 11 device to use for interoperability in this thread.
- cudaError_t cudaGraphicsD3D11RegisterResource (struct cudaGraphicsResource **resource, ID3D11Resource **pD3DResource, unsigned int flags)
 Register a Direct3D 11 resource for access by CUDA.

4.15.1 Detailed Description

This section describes the Direct3D 11 interoperability functions of the CUDA runtime application programming interface.

4.15.2 Function Documentation

4.15.2.1 cudaError_t cudaD3D11GetDevice (int * device, IDXGIAdapter * pAdapter)

Returns in *device the CUDA-compatible device corresponding to the adapter pAdapter obtained from IDXGI-Factory::EnumAdapters. This call will succeed only if a device on adapter pAdapter is Cuda-compatible.

Parameters:

```
device - Returns the device corresponding to pAdapterpAdapter - D3D11 adapter to get device for
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda Graphics Unregister Resource, \quad cuda Graphics Map Resources, \quad cuda Graphics Sub Resource Get Mapped Array, \\ cuda Graphics Resource Get Mapped Pointer$

4.15.2.2 cudaError_t cudaD3D11SetDirect3DDevice (ID3D11Device * pD3D11Device)

Records pD3D11Device as the Direct3D 11 device to use for Direct3D 11 interoperability on this host thread. In order to use Direct3D 11 interoperability, this call must be made before any non-device management CUDA runtime calls on this thread. In that case, this call will return cudaErrorSetOnActiveProcess.

Successful context creation on pD3D11Device will increase the internal reference count on pD3D11Device. This reference count will be decremented upon destruction of this context through cudaThreadExit().

Parameters:

pD3D11Device - Direct3D device to use for interoperability

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorInvalidValue, cudaErrorSetOnActiveProcess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaD3D11GetDevice, cudaGraphicsD3D11RegisterResource

4.15.2.3 cudaError_t cudaGraphicsD3D11RegisterResource (struct cudaGraphicsResource ** resource, ID3D11Resource ** pD3DResource*, unsigned int flags)

Registers the Direct3D 11 resource pD3DResource for access by CUDA.

If this call is successful, then the application will be able to map and unmap this resource until it is unregistered through cudaGraphicsUnregisterResource(). Also on success, this call will increase the internal reference count on pD3DResource. This reference count will be decremented when this resource is unregistered through cudaGraphicsUnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pD3DResource must be one of the following.

- ID3D11Buffer: may be accessed via a device pointer
- ID3D11Texture1D: individual subresources of the texture may be accessed via arrays
- ID3D11Texture2D: individual subresources of the texture may be accessed via arrays
- ID3D11Texture3D: individual subresources of the texture may be accessed via arrays

The flags argument may be used to specify additional parameters at register time. The only valid value for this parameter is

• cudaGraphicsRegisterFlagsNone

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized using cudaD3D11SetDirect3DDevice then cudaErrorInvalidDevice is returned. If pD3DResource is of incorrect type or is already registered, then cudaErrorInvalidResourceHandle is returned. If pD3DResource cannot be registered, then cudaErrorUnknown is returned.

Parameters:

resource - Pointer to returned resource handlepD3DResource - Direct3D resource to registerflags - Parameters for resource registration

Returns:

cudaSuccess, cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda D3D11Set Direct 3DDevice\ cuda Graphics Unregister Resource,\ cuda Graphics Map Resources,\ cuda Graphics Sub Resource Get Mapped Pointer$

4.16 Graphics Interoperability

Functions

cudaError_t cudaGraphicsMapResources (int count, struct cudaGraphicsResource **resources, cudaStream_t stream)

Map graphics resources for access by CUDA.

cudaError_t cudaGraphicsResourceGetMappedPointer (void **devPtr, size_t *size, struct cudaGraphicsResource *resource)

Get an device pointer through which to access a mapped graphics resource.

- cudaError_t cudaGraphicsResourceSetMapFlags (struct cudaGraphicsResource *resource, unsigned int flags)

 Set usage flags for mapping a graphics resource.
- cudaError_t cudaGraphicsSubResourceGetMappedArray (cudaArray **array, struct cudaGraphicsResource *resource, unsigned int arrayIndex, unsigned int mipLevel)

Get an array through which to access a subresource of a mapped graphics resource.

• cudaError_t cudaGraphicsUnmapResources (int count, struct cudaGraphicsResource **resources, cudaStream_t stream)

Unmap graphics resources.

cudaError_t cudaGraphicsUnregisterResource (struct cudaGraphicsResource *resource)

Unregisters a graphics resource for access by CUDA.

4.16.1 Detailed Description

This section describes the graphics interoperability functions of the CUDA runtime application programming interface.

4.16.2 Function Documentation

4.16.2.1 cudaError_t cudaGraphicsMapResources (int *count*, struct cudaGraphicsResource ** *resources*, cudaStream_t *stream*)

Maps the count graphics resources in resources for access by CUDA.

The resources in resources may be accessed by CUDA until they are unmapped. The graphics API from which resources were registered should not access any resources while they are mapped by CUDA. If an application does so, the results are undefined.

This function provides the synchronization guarantee that any graphics calls issued before cudaGraphicsMapResources() will complete before any subsequent CUDA work issued in stream begins.

If resources contains any duplicate entries then cudaErrorInvalidResourceHandle is returned. If any of resources are presently mapped for access by CUDA then cudaErrorUnknown is returned.

Parameters:

count - Number of resources to mapresources - Resources to map for CUDA

stream - Stream for synchronization

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuda Graphics Resource Get Mapped Pointer\ cuda Graphics Sub Resource Get Mapped Array\ cuda Graphics Unmap Resources$

4.16.2.2 cudaError_t cudaGraphicsResourceGetMappedPointer (void ** devPtr, size_t * size, struct cudaGraphicsResource * resource)

Returns in *devPtr a pointer through which the mapped graphics resource resource may be accessed. Returns in *size the size of the memory in bytes which may be accessed from that pointer. The value set in devPtr may change every time that resource is mapped.

If resource is not a buffer then it cannot be accessed via a pointer and cudaErrorUnknown is returned. If resource is not mapped then cudaErrorUnknown is returned. *

Parameters:

```
devPtr - Returned pointer through which resource may be accessedsize - Returned size of the buffer accessible starting at *devPtrresource - Mapped resource to access
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Map Resources, cuda Graphics Sub Resource Get Mapped Array

4.16.2.3 cudaError_t cudaGraphicsResourceSetMapFlags (struct cudaGraphicsResource * resource, unsigned int flags)

Set flags for mapping the graphics resource resource.

Changes to flags will take effect the next time resource is mapped. The flags argument may be any of the following:

- cudaGraphicsMapFlagsNone: Specifies no hints about how resource will be used. It is therefore assumed that CUDA may read from or write to resource.
- cudaGraphicsMapFlagsReadOnly: Specifies that CUDA will not write to resource.

• cudaGraphicsMapFlagsWriteDiscard: Specifies CUDA will not read from resource and will write over the entire contents of resource, so none of the data previously stored in resource will be preserved.

If resource is presently mapped for access by CUDA then cudaErrorUnknown is returned. If flags is not one of the above values then cudaErrorInvalidValue is returned.

Parameters:

```
resource - Registered resource to set flags forflags - Parameters for resource mapping
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsMapResources

4.16.2.4 cudaError_t cudaGraphicsSubResourceGetMappedArray (cudaArray ** array, struct cudaGraphicsResource * resource, unsigned int arrayIndex, unsigned int mipLevel)

Returns in *array an array through which the subresource of the mapped graphics resource resource which corresponds to array index arrayIndex and mipmap level mipLevel may be accessed. The value set in array may change every time that resource is mapped.

If resource is not a texture then it cannot be accessed via an array and cudaErrorUnknown is returned. If arrayIndex is not a valid array index for resource then cudaErrorInvalidValue is returned. If mipLevel is not a valid mipmap level for resource then cudaErrorInvalidValue is returned. If resource is not mapped then cudaErrorUnknown is returned.

Parameters:

```
array - Returned array through which a subresource of resource may be accessed
```

resource - Mapped resource to access

arrayIndex - Array index for array textures or cubemap face index as defined by cudaGraphicsCubeFace for cubemap textures for the subresource to access

mipLevel - Mipmap level for the subresource to access

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuda Graphics Resource Get Mapped Pointer

4.16.2.5 cudaError_t cudaGraphicsUnmapResources (int *count*, struct cudaGraphicsResource ** *resources*, cudaStream t *stream*)

Unmaps the count graphics resources in resources.

Once unmapped, the resources in resources may not be accessed by CUDA until they are mapped again.

This function provides the synchronization guarantee that any CUDA work issued in stream before cudaGraphicsUnmapResources() will complete before any subsequently issued graphics work begins.

If resources contains any duplicate entries then cudaErrorInvalidResourceHandle is returned. If any of resources are not presently mapped for access by Cuda then cudaErrorUnknown is returned.

Parameters:

```
count - Number of resources to unmapresources - Resources to unmapstream - Stream for synchronization
```

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsMapResources

4.16.2.6 cudaError_t cudaGraphicsUnregisterResource (struct cudaGraphicsResource * resource)

Unregisters the graphics resource resource so it is not accessible by CUDA unless registered again.

If resource is invalid then cudaErrorInvalidResourceHandle is returned.

Parameters:

```
resource - Resource to unregister
```

Returns:

cudaSuccess, cudaErrorInvalidResourceHandle, cudaErrorUnknown

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGraphicsD3D9RegisterResource, cudaGraphicsD3D10RegisterResource, cudaGraphicsD3D11RegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsGLRegisterImage

4.17 Texture Reference Management

Functions

• cudaError_t cudaBindTexture (size_t *offset, const struct textureReference *texref, const void *devPtr, const struct cudaChannelFormatDesc *desc, size_t size)

Binds a memory area to a texture.

• cudaError_t cudaBindTexture2D (size_t *offset, const struct textureReference *texref, const void *devPtr, const struct cudaChannelFormatDesc *desc, size_t width, size_t height, size_t pitch)

Binds a 2D memory area to a texture.

• cudaError_t cudaBindTextureToArray (const struct textureReference *texref, const struct cudaArray *array, const struct cudaChannelFormatDesc *desc)

Binds an array to a texture.

struct cudaChannelFormatDesc cudaCreateChannelDesc (int x, int y, int z, int w, enum cudaChannelFormatKind f)

Returns a channel descriptor using the specified format.

- cudaError_t cudaGetChannelDesc (struct cudaChannelFormatDesc *desc, const struct cudaArray *array)

 Get the channel descriptor of an array.
- cudaError_t cudaGetTextureAlignmentOffset (size_t *offset, const struct textureReference *texref)

 Get the alignment offset of a texture.
- cudaError_t cudaGetTextureReference (const struct textureReference **texref, const char *symbol)

 Get the texture reference associated with a symbol.
- cudaError_t cudaUnbindTexture (const struct textureReference *texref)

 Unbinds a texture.

4.17.1 Detailed Description

This section describes the low level texture reference management functions of the CUDA runtime application programming interface.

4.17.2 Function Documentation

4.17.2.1 cudaError_t cudaBindTexture (size_t * offset, const struct textureReference * texref, const void * devPtr, const struct cudaChannelFormatDesc * desc, size_t size)

Binds size bytes of the memory area pointed to by devPtr to the texture reference texref. desc describes how the memory is interpreted when fetching values from the texture. Any memory previously bound to texref is unbound.

Since the hardware enforces an alignment requirement on texture base addresses, cudaBindTexture() returns in *offset a byte offset that must be applied to texture fetches in order to read from the desired memory. This offset must be divided by the texel size and passed to kernels that read from the texture so they can be applied to the

tex1Dfetch() function. If the device memory pointer was returned from cudaMalloc(), the offset is guaranteed to be 0 and NULL may be passed as the offset parameter.

Parameters:

```
offset - Offset in bytes
texref - Texture to bind
devPtr - Memory area on device
desc - Channel format
size - Size of the memory area pointed to by devPtr
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C++ API), cudaBindTexture2D (C API), cudaBindTextureToArray (C API), cudaUnbindTexture (C API), cudaGetTexture-AlignmentOffset (C API)

4.17.2.2 cudaError_t cudaBindTexture2D (size_t * offset, const struct textureReference * texref, const void * devPtr, const struct cudaChannelFormatDesc * desc, size_t width, size_t height, size_t pitch)

Binds the 2D memory area pointed to by devPtr to the texture reference texref. The size of the area is constrained by width in texel units, height in texel units, and pitch in byte units. desc describes how the memory is interpreted when fetching values from the texture. Any memory previously bound to texref is unbound.

Since the hardware enforces an alignment requirement on texture base addresses, cudaBindTexture2D() returns in *offset a byte offset that must be applied to texture fetches in order to read from the desired memory. This offset must be divided by the texel size and passed to kernels that read from the texture so they can be applied to the tex2D() function. If the device memory pointer was returned from cudaMalloc(), the offset is guaranteed to be 0 and NULL may be passed as the offset parameter.

Parameters:

```
offset - Offset in bytes
texref - Texture reference to bind
devPtr - 2D memory area on device
desc - Channel format
width - Width in texel units
height - Height in texel units
pitch - Pitch in bytes
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C API), cudaBindTextureToArray (C API), cudaGetTextureAlignmentOffset (C API)

4.17.2.3 cudaError_t cudaBindTextureToArray (const struct textureReference * texref, const struct cudaArray * array, const struct cudaChannelFormatDesc * desc)

Binds the CUDA array array to the texture reference texref. desc describes how the memory is interpreted when fetching values from the texture. Any CUDA array previously bound to texref is unbound.

Parameters:

```
texref - Texture to bindarray - Memory array on devicedesc - Channel format
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture2D (C API), cudaBindTextureToArray (C++ API), cudaUnbindTexture (C API), cudaGetTextureAlignmentOffset (C API)

4.17.2.4 struct cudaChannelFormatDesc cudaCreateChannelDesc (int x, int y, int z, int w, enum cudaChannelFormatKind f) [read]

Returns a channel descriptor with format f and number of bits of each component x, y, z, and w. The cudaChannelFormatDesc is defined as:

```
struct cudaChannelFormatDesc {
  int x, y, z, w;
  enum cudaChannelFormatKind f;
};
```

where cudaChannelFormatKind is one of cudaChannelFormatKindSigned, cudaChannelFormatKindUnsigned, or cudaChannelFormatKindFloat.

Parameters:

```
x - X component
```

y - Y component

```
z - Z component
```

w - W component

f - Channel format

Returns:

Channel descriptor with format f

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture2D (C API), cudaBindTextureToArray (C API), cudaUnbindTexture (C API), cudaGetTexture-AlignmentOffset (C API)

4.17.2.5 cudaError_t cudaGetChannelDesc (struct cudaChannelFormatDesc * *desc*, const struct cudaArray * *array*)

Returns in *desc the channel descriptor of the CUDA array array.

Parameters:

```
desc - Channel formatarray - Memory array on device
```

Returns:

cudaSuccess, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture2D (C API), cudaBindTextureToArray (C API), cudaUnbindTexture (C API), cudaGetTextureAlignmentOffset (C API)

4.17.2.6 cudaError_t cudaGetTextureAlignmentOffset (size_t * offset, const struct textureReference * textef)

Returns in *offset the offset that was returned when texture reference texref was bound.

Parameters:

```
offset - Offset of texture reference in bytestexref - Texture to get offset of
```

Returns:

cudaSuccess, cudaErrorInvalidTexture, cudaErrorInvalidTextureBinding

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture2D (C API), cudaBindTextureToArray (C API), cudaUnbindTexture (C API), cudaGetTexture-AlignmentOffset (C++ API)

4.17.2.7 cudaError_t cudaGetTextureReference (const struct textureReference ** textef, const char * symbol)

Returns in *texref the structure associated to the texture reference defined by symbol symbol.

Parameters:

```
texref - Texture associated with symbolsymbol - Symbol to find texture reference for
```

Returns:

cudaSuccess, cudaErrorInvalidTexture,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetChannelDesc, cudaGetTextureAlignmentOffset (C API), cudaBindTexture (C API), cudaBindTexture2D (C API), cudaBindTextureToArray (C API), cudaUnbindTexture (C API)

4.17.2.8 cudaError_t cudaUnbindTexture (const struct textureReference * textef)

Unbinds the texture bound to texref.

Parameters:

texref - Texture to unbind

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture2D (C API), cudaBindTextureToArray (C API), cudaUnbindTexture (C++ API), cudaGetTextureAlignmentOffset (C API)

4.18 Version Management

Functions

cudaError_t cudaDriverGetVersion (int *driverVersion)

Returns the CUDA driver version.

• cudaError_t cudaRuntimeGetVersion (int *runtimeVersion)

Returns the CUDA Runtime version.

4.18.1 Function Documentation

4.18.1.1 cudaError_t cudaDriverGetVersion (int * driverVersion)

Returns in *driverVersion the version number of the installed CUDA driver. If no driver is installed, then 0 is returned as the driver version (via driverVersion). This function automatically returns cudaErrorInvalidValue if the driverVersion argument is NULL.

Parameters:

driver Version - Returns the CUDA driver version.

Returns:

cudaSuccess, cudaErrorInvalidValue

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaRuntimeGetVersion

4.18.1.2 cudaError_t cudaRuntimeGetVersion (int * runtimeVersion)

Returns in *runtimeVersion the version number of the installed CUDA Runtime. This function automatically returns cudaErrorInvalidValue if the runtimeVersion argument is NULL.

Parameters:

runtime Version - Returns the CUDA Runtime version.

Returns:

cudaSuccess, cudaErrorInvalidValue

See also:

cudaDriverGetVersion

4.19 C++ API Routines 103

4.19 C++ API Routines

C++-style interface built on top of CUDA runtime API.

Functions

template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTexture (size_t *offset, const struct texture < T, dim, readMode > &tex, const void *devPtr, size_t size=UINT_MAX)

```
[C++ API] Binds a memory area to a texture
```

template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTexture (size_t *offset, const struct texture < T, dim, readMode > &tex, const void *devPtr, const struct cudaChannelFormatDesc &desc, size_t size=UINT_MAX)

```
[C++ API] Binds a memory area to a texture
```

• template<class T, int dim, enum cudaTextureReadMode readMode> cudaError_t cudaBindTexture2D (size_t *offset, const struct texture< T, dim, readMode > &tex, const void *devPtr, const struct cudaChannelFormatDesc &desc, size_t width, size_t height, size_t pitch)

```
[C++ API] Binds a 2D memory area to a texture
```

template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTextureToArray (const struct texture < T, dim, readMode > &tex, const struct cudaArray *array)

```
[C++ API] Binds an array to a texture
```

template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTextureToArray (const struct texture < T, dim, readMode > &tex, const struct cudaArray *array, const struct cudaChannelFormatDesc &desc)

```
[C++ API] Binds an array to a texture
```

template < class T >
 cudaChannelFormatDesc cudaCreateChannelDesc (void)

[C++ API] Returns a channel descriptor using the specified format

template < class T >
 cudaError_t cudaFuncGetAttributes (struct cudaFuncAttributes *attr, T *entry)

[C++ API] Find out attributes for a given function

• template < class T > cudaError_t cudaFuncSetCacheConfig (T *func, enum cudaFuncCache cacheConfig)

Sets the preferred cache configuration for a device function.

template < class T >
 cudaError_t cudaGetSymbolAddress (void **devPtr, const T & symbol)
 [C++ API] Finds the address associated with a CUDA symbol

template < class T >
 cudaError_t cudaGetSymbolSize (size_t *size, const T &symbol)
 [C++ API] Finds the size of the object associated with a CUDA symbol

```
    template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaGetTextureAlignmentOffset (size_t *offset, const struct texture < T, dim, readMode > &tex)
        [C++ API] Get the alignment offset of a texture
    template < class T > cudaError_t cudaLaunch (T *entry)
        [C++ API] Launches a device function
    template < class T > cudaError_t cudaSetupArgument (T arg, size_t offset)
        [C++ API] Configure a device launch
    template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaUnbindTexture (const struct texture < T, dim, readMode > &tex)
        [C++ API] Unbinds a texture
```

4.19.1 Detailed Description

This section describes the C++ high level API functions of the CUDA runtime application programming interface. To use these functions, your application needs to be compiled with the nvcc compiler.

4.19.2 Function Documentation

4.19.2.1 template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTexture (size_t * offset, const struct texture < T, dim, readMode > & tex, const void * devPtr, size_t size = UINT_MAX)

Binds size bytes of the memory area pointed to by devPtr to texture reference tex. The channel descriptor is inherited from the texture reference type. The offset parameter is an optional byte offset as with the low-level cudaBindTexture(size_t*, const struct textureReference*, const void*, const struct cudaChannelFormatDesc*, size_t) function. Any memory previously bound to tex is unbound.

Parameters:

```
offset - Offset in bytes
tex - Texture to bind
devPtr - Memory area on device
size - Size of the memory area pointed to by devPtr
```

Returns:

cuda Success, cuda Error Invalid Value, cuda Error Invalid Device Pointer, cuda Error Invalid Texture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture (C++ API, inherited channel descriptor), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C++ API, inherited channel descriptor), cudaUnbindTexture (C++ API), cudaGetTextureAlignmentOffset (C++ API)

4.19 C++ API Routines 105

4.19.2.2 template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTexture (size_t * offset, const struct texture < T, dim, readMode > & tex, const void * devPtr, const struct cudaChannelFormatDesc & desc, size t size = UINT MAX)

Binds size bytes of the memory area pointed to by devPtr to texture reference tex. desc describes how the memory is interpreted when fetching values from the texture. The offset parameter is an optional byte offset as with the low-level cudaBindTexture() function. Any memory previously bound to tex is unbound.

Parameters:

offset - Offset in bytes
tex - Texture to bind
devPtr - Memory area on device
desc - Channel format
size - Size of the memory area pointed to by devPtr

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C API), cudaBindTexture (C++ API, inherited channel descriptor), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C++ API, inherited channel descriptor), cudaUnbindTexture (C++ API), cudaGetTextureAlignmentOffset (C++ API)

4.19.2.3 template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaBindTexture2D (size_t * offset, const struct texture < T, dim, readMode > & tex, const void * devPtr, const struct cudaChannelFormatDesc & desc, size_t width, size_t height, size_t pitch)

Binds the 2D memory area pointed to by devPtr to the texture reference tex. The size of the area is constrained by width in texel units, height in texel units, and pitch in byte units. desc describes how the memory is interpreted when fetching values from the texture. Any memory previously bound to tex is unbound.

Since the hardware enforces an alignment requirement on texture base addresses, cudaBindTexture2D() returns in *offset a byte offset that must be applied to texture fetches in order to read from the desired memory. This offset must be divided by the texel size and passed to kernels that read from the texture so they can be applied to the tex2D() function. If the device memory pointer was returned from cudaMalloc(), the offset is guaranteed to be 0 and NULL may be passed as the offset parameter.

Parameters:

offset - Offset in bytes
tex - Texture reference to bind
devPtr - 2D memory area on device
desc - Channel format
width - Width in texel units

```
height - Height in texel unitspitch - Pitch in bytes
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C++ API), cudaBindTexture (C++ API, inherited channel descriptor), cudaBindTexture2D (C API), cudaBindTextureToArray (C++ API), cudaBindTextureToArray (C++ API, inherited channel descriptor), cudaUnbindTexture (C++ API), cudaGetTextureAlignmentOffset (C++ API)

4.19.2.4 template < class T , int dim, enum cuda Texture Read Mode > cuda Error_t cuda Bind Texture To Array (const struct texture < T, dim, read Mode > & tex, const struct cuda Array * array)

Binds the CUDA array array to the texture reference tex. The channel descriptor is inherited from the CUDA array. Any CUDA array previously bound to tex is unbound.

Parameters:

```
tex - Texture to bindarray - Memory array on device
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C++ API), cudaBindTexture (C++ API), cudaBindTexture2D (C++ API), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C API), cudaBindTextureToArray (C++ API), cudaUnbindTexture (C++ API), cudaGetTexture-AlignmentOffset (C++ API)

4.19.2.5 template < class T , int dim, enum cuda Texture Read Mode > cuda Error_t cuda Bind Texture To Array (const struct texture < T, dim, read Mode > & tex, const struct cuda Array * array, const struct cuda Channel Format Desc & desc)

Binds the CUDA array array to the texture reference tex. desc describes how the memory is interpreted when fetching values from the texture. Any CUDA array previously bound to tex is unbound.

Parameters:

tex - Texture to bind

4.19 C++ API Routines 107

```
array - Memory array on devicedesc - Channel format
```

Returns:

cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer, cudaErrorInvalidTexture

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C++ API), cudaBindTexture (C++ API, inherited channel descriptor), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C API), cudaBindTextureToArray (C++ API, inherited channel descriptor), cudaUnbindTexture (C++ API), cudaGetTextureAlignmentOffset (C++ API)

4.19.2.6 template < class T > cudaChannelFormatDesc cudaCreateChannelDesc (void)

Returns a channel descriptor with format f and number of bits of each component x, y, z, and w. The cudaChannelFormatDesc is defined as:

```
struct cudaChannelFormatDesc {
  int x, y, z, w;
  enum cudaChannelFormatKind f;
};
```

where cudaChannelFormatKind is one of cudaChannelFormatKindSigned, cudaChannelFormatKindUnsigned, or cudaChannelFormatKindFloat.

Returns:

Channel descriptor with format f

See also:

cudaCreateChannelDesc (Low level), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (High level), cudaBindTexture (High level), cudaBindTexture2D (High level), cudaBindTextureToArray (High level), cudaBindTextureToArray (High level), cudaUnbindTexture (High level), cudaGetTextureAlignmentOffset (High level)

4.19.2.7 template < class T > cudaError_t cudaFuncGetAttributes (struct cudaFuncAttributes * attr, T * entry)

This function obtains the attributes of a function specified via entry. The parameter entry can either be a pointer to a function that executes on the device, or it can be a character string specifying the fully-decorated (C++) name of a function that executes on the device. The parameter specified by entry must be declared as a __global__ function. The fetched attributes are placed in attr. If the specified function does not exist, then cudaErrorInvalidDeviceFunction is returned.

Parameters:

attr - Return pointer to function's attributes

entry - Function to get attributes of

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorInvalidDeviceFunction

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C++ API), cudaFuncGetAttributes (C API), cudaLaunch (C++ API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C++ API)

4.19.2.8 template < class T > cudaError_t cudaFuncSetCacheConfig (T * func, enum cudaFuncCache cacheConfig)

On devices where the L1 cache and shared memory use the same hardware resources, this sets through cacheConfig the preferred cache configuration for the function specified via func. This is only a preference. The runtime will use the requested configuration if possible, but it is free to choose a different configuration if required to execute func.

func can either be a pointer to a function that executes on the device, or it can be a character string specifying the fully-decorated (C++) name for a function that executes on the device. The parameter specified by func must be declared as a __global__ function. If the specified function does not exist, then cudaErrorInvalidDeviceFunction is returned.

This setting does nothing on devices where the size of the L1 cache and shared memory are fixed.

Switching between configuration modes may insert a device-side synchronization point for streamed kernel launches.

Parameters:

```
func - Device char string naming device functioncacheConfig - Cache configuration mode
```

Returns:

cudaSuccess, cudaErrorInitializationError, cudaErrorInvalidDeviceFunction

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C API), cudaFuncGetAttributes (C++ API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C++ API)

4.19.2.9 template < class T > cudaError_t cudaGetSymbolAddress (void ** devPtr, const T & symbol)

Returns in *devPtr the address of symbol symbol on the device. symbol can either be a variable that resides in global memory space, or it can be a character string, naming a variable that resides in global memory space. If symbol cannot be found, or if symbol is not declared in the global memory space, *devPtr is unchanged and the error cudaErrorInvalidSymbol is returned.

4.19 C++ API Routines 109

Parameters:

```
devPtr - Return device pointer associated with symbolsymbol - Global variable or string symbol to search for
```

Returns:

cudaSuccess, cudaErrorInvalidSymbol, cudaErrorAddressOfConstant

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaGetSymbolAddress (C API) cudaGetSymbolSize (C++ API)

4.19.2.10 template < class T > cudaError_t cudaGetSymbolSize (size_t * size, const T & symbol)

Returns in *size the size of symbol symbol. symbol can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. If symbol cannot be found, or if symbol is not declared in global or constant memory space, *size is unchanged and the error cudaErrorInvalidSymbol is returned.

Parameters:

```
size - Size of object associated with symbolsymbol - Global variable or string symbol to find size of
```

Returns:

cudaSuccess, cudaErrorInvalidSymbol

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

```
cudaGetSymbolAddress (C++ API) cudaGetSymbolSize (C API)
```

$\label{lem:total_def} \textbf{4.19.2.11} \quad template < class \ T \ , int \ dim, \ enum \ cuda Texture Read Mode \ read Mode > cuda Error_t \\ cuda Get Texture A lignment Offset \ (size_t * \textit{offset}, \ const \ struct \ texture < T, \ dim, \ read Mode > \& \textit{tex}) \\$

Returns in *offset the offset that was returned when texture reference tex was bound.

Parameters:

```
offset - Offset of texture reference in bytestex - Texture to get offset of
```

Returns:

cudaSuccess, cudaErrorInvalidTexture, cudaErrorInvalidTextureBinding

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C++ API), cudaBindTexture (C++ API), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C++ API), cudaBindTextureToArray (C++ API), cudaBindTextureToArray (C++ API), cudaGetTextureAlignmentOffset (C API)

4.19.2.12 template < class T > cudaError_t cudaLaunch (T * *entry*)

Launches the function entry on the device. The parameter entry can either be a function that executes on the device, or it can be a character string, naming a function that executes on the device. The parameter specified by entry must be declared as a __global__ function. cudaLaunch() must be preceded by a call to cudaConfigureCall() since it pops the data that was pushed by cudaConfigureCall() from the execution stack.

Parameters:

entry - Device function pointer or char string naming device function to execute

Returns:

cudaSuccess, cudaErrorInvalidDeviceFunction, cudaErrorInvalidConfiguration, cudaErrorLaunchFailure, cudaErrorLaunchOutOfResources

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncSetCacheConfig (C++ API), cudaFuncGetAttributes (C++ API), cudaLaunch (C API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C++ API)

4.19.2.13 template < class T > cudaError_t cudaSetupArgument (T arg, size_t offset)

Pushes size bytes of the argument pointed to by arg at offset bytes from the start of the parameter passing area, which starts at offset 0. The arguments are stored in the top of the execution stack. cudaSetupArgument() must be preceded by a call to cudaConfigureCall().

Parameters:

```
arg - Argument to push for a kernel launchoffset - Offset in argument stack to push new arg
```

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaConfigureCall, cudaFuncGetAttributes (C++ API), cudaLaunch (C++ API), cudaSetDoubleForDevice, cudaSetDoubleForHost, cudaSetupArgument (C API)

4.19 C++ API Routines 111

4.19.2.14 template < class T , int dim, enum cudaTextureReadMode readMode > cudaError_t cudaUnbindTexture (const struct texture < T, dim, readMode > & tex)

Unbinds the texture bound to tex.

Parameters:

tex - Texture to unbind

Returns:

cudaSuccess

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cudaCreateChannelDesc (C++ API), cudaGetChannelDesc, cudaGetTextureReference, cudaBindTexture (C++ API), cudaBindTexture (C++ API), cudaBindTexture2D (C++ API), cudaBindTextureToArray (C++ API), cudaBindTextureToArray (C++ API), cudaBindTextureToArray (C++ API), cudaGetTextureAlignmentOffset (C++ API)

4.20 Data types used by CUDA Runtime

Data Structures

- struct cudaChannelFormatDesc
- struct cudaDeviceProp
- struct cudaExtent
- struct cudaFuncAttributes
- struct cudaMemcpy3DParms
- struct cudaPitchedPtr
- struct cudaPos

Data types used by CUDA Runtime

 enum cudaChannelFormatKind { cudaChannelFormatKindSigned,

Data types used by CUDA Runtime

Author:

NVIDIA Corporation

```
cudaChannelFormatKindUnsigned, cudaChannelFormatKindFloat, cudaChannelFormatKindNone }
• enum cudaComputeMode { cudaComputeModeDefault, cudaComputeModeExclusive, cudaComputeModeProhibited }
• enum cudaError { cudaSuccess, cudaErrorMissingConfiguration, cudaErrorMemoryAllocation, cudaErrorInitializationError, cudaErrorLaunchFailure, cudaErrorPriorLaunchFailure, cudaErrorLaunchTimeout,
```

cudaErrorLaunchOutOfResources, cudaErrorInvalidDeviceFunction, cudaErrorInvalidConfiguration,

cudaErrorMapBufferObjectFailed,

cudaErrorInvalidDevice, cudaErrorInvalidValue, cudaErrorInvalidPitchValue, cudaErrorInvalidSymbol,

```
cudaErrorUnmapBufferObjectFailed,
 cudaErrorInvalidHostPointer,
 cudaErrorInvalidDevicePointer,
 cudaErrorInvalidTexture,
 cudaErrorInvalidTextureBinding,
 cudaErrorInvalidChannelDescriptor,
 cudaErrorInvalidMemcpyDirection,
 cudaErrorAddressOfConstant,
 cudaErrorTextureFetchFailed.
 cudaErrorTextureNotBound,
 cudaErrorSynchronizationError,
 cudaErrorInvalidFilterSetting,
 cudaErrorInvalidNormSetting,
 cudaErrorMixedDeviceExecution,
 cudaErrorCudartUnloading,
 cudaErrorUnknown.
 cudaErrorNotYetImplemented,
 cudaErrorMemoryValueTooLarge,
 cudaErrorInvalidResourceHandle,
 cudaErrorNotReady,
 cudaErrorInsufficientDriver,
 cudaErrorSetOnActiveProcess,
 cudaErrorNoDevice,
 cudaErrorECCUncorrectable.
 cudaErrorStartupFailure,
 cudaErrorApiFailureBase }
enum cudaFuncCache {
 cudaFuncCachePreferNone,
 cudaFuncCachePreferShared,
 cudaFuncCachePreferL1 }
enum cudaGraphicsCubeFace {
 cudaGraphicsCubeFacePositiveX,
 cudaGraphicsCubeFaceNegativeX,
 cudaGraphicsCubeFacePositiveY,
 cudaGraphicsCubeFaceNegativeY,
 cudaGraphicsCubeFacePositiveZ,
 cudaGraphicsCubeFaceNegativeZ }
enum cudaGraphicsMapFlags {
 cudaGraphicsMapFlagsNone,
 cudaGraphicsMapFlagsReadOnly,
 cudaGraphicsMapFlagsWriteDiscard }
• enum cudaGraphicsRegisterFlags { cudaGraphicsRegisterFlagsNone }
```

enum cudaMemcpyKind {
 cudaMemcpyHostToHost,
 cudaMemcpyHostToDevice,
 cudaMemcpyDeviceToHost,
 cudaMemcpyDeviceToDevice }

- typedef enum cudaError cudaError_t
- typedef int cudaEvent_t
- typedef int cudaStream_t
- #define cudaDeviceBlockingSync

Device flag - Use blocking synchronization.

• #define cudaDeviceLmemResizeToMax

Device flag - Keep local memory allocation after launch.

• #define cudaDeviceMapHost

Device flag - Support mapped pinned allocations.

• #define cudaDeviceMask

Device flags mask.

• #define cudaDevicePropDontCare

Empty device properties.

• #define cudaDeviceScheduleAuto

Device flag - Automatic scheduling.

• #define cudaDeviceScheduleSpin

Device flag - Spin default scheduling.

• #define cudaDeviceScheduleYield

Device flag - Yield default scheduling.

• #define cudaEventBlockingSync

Event uses blocking synchronization.

• #define cudaEventDefault

Default event flag.

• #define cudaHostAllocDefault

Default page-locked allocation flag.

#define cudaHostAllocMapped

Map allocation into device space.

• #define cudaHostAllocPortable

Pinned memory accessible by all CUDA contexts.

• #define cudaHostAllocWriteCombined

Write-combined memory.

4.20.1 Typedef Documentation

4.20.1.1 typedef enum cudaError cudaError_t

CUDA Error types

4.20.1.2 typedef int cudaEvent_t

CUDA event types

4.20.1.3 typedef int cudaStream_t

CUDA stream

4.20.2 Enumeration Type Documentation

4.20.2.1 enum cudaChannelFormatKind

Channel format kind

Enumerator:

```
    cudaChannelFormatKindSigned Signed channel format.
    cudaChannelFormatKindUnsigned Unsigned channel format.
    cudaChannelFormatKindFloat Float channel format.
    cudaChannelFormatKindNone No channel format.
```

4.20.2.2 enum cudaComputeMode

CUDA device compute modes

Enumerator:

cudaComputeModeDefault Default compute mode (Multiple threads can use cudaSetDevice() with this device).

cudaComputeModeExclusive Compute-exclusive mode (Only one thread will be able to use cudaSetDevice() with this device).

cudaComputeModeProhibited Compute-prohibited mode (No threads can use cudaSetDevice() with this device).

4.20.2.3 enum cudaError

CUDA error types

Enumerator:

```
cudaSuccess No errors.cudaErrorMissingConfiguration Missing configuration error.cudaErrorMemoryAllocation Memory allocation error.
```

cudaErrorInitializationError Initialization error. cudaErrorLaunchFailure Launch failure. cudaErrorPriorLaunchFailure Prior launch failure. cudaErrorLaunchTimeout Launch timeout error. cudaErrorLaunchOutOfResources Launch out of resources error. cudaErrorInvalidDeviceFunction Invalid device function. cudaErrorInvalidConfiguration Invalid configuration. cudaErrorInvalidDevice Invalid device. cudaErrorInvalidValue Invalid value. cudaErrorInvalidPitchValue Invalid pitch value. cudaErrorInvalidSymbol Invalid symbol. cudaErrorMapBufferObjectFailed Map buffer object failed. cudaErrorUnmapBufferObjectFailed Unmap buffer object failed. cudaErrorInvalidHostPointer Invalid host pointer. cudaErrorInvalidDevicePointer Invalid device pointer. cudaErrorInvalidTexture Invalid texture. cudaErrorInvalidTextureBinding Invalid texture binding. cudaErrorInvalidChannelDescriptor Invalid channel descriptor. cudaErrorInvalidMemcpyDirection Invalid memcpy direction. cudaErrorAddressOfConstant Address of constant error. cudaErrorTextureFetchFailed Texture fetch failed. cudaErrorTextureNotBound Texture not bound error. cudaErrorSynchronizationError Synchronization error. cudaErrorInvalidFilterSetting Invalid filter setting. cudaErrorInvalidNormSetting Invalid norm setting. cudaErrorMixedDeviceExecution Mixed device execution. cudaErrorCudartUnloading CUDA runtime unloading. cudaErrorUnknown Unknown error condition. cudaErrorNotYetImplemented Function not yet implemented. cudaErrorMemoryValueTooLarge Memory value too large. cudaErrorInvalidResourceHandle Invalid resource handle. cudaErrorNotReady Not ready error. cudaErrorInsufficientDriver CUDA runtime is newer than driver. cudaErrorSetOnActiveProcess Set on active process error. cudaErrorNoDevice No available CUDA device. cudaErrorECCUncorrectable Uncorrectable ECC error detected. cudaErrorStartupFailure Startup failure. cudaErrorApiFailureBase API failure base.

4.20.2.4 enum cudaFuncCache

CUDA function cache configurations

Enumerator:

```
    cudaFuncCachePreferNone Default function cache configuration, no preference.
    cudaFuncCachePreferShared Prefer larger shared memory and smaller L1 cache.
    cudaFuncCachePreferL1 Prefer larger L1 cache and smaller shared memory.
```

4.20.2.5 enum cudaGraphicsCubeFace

CUDA graphics interop array indices for cube maps

Enumerator:

```
    cudaGraphicsCubeFacePositiveX
    Positive X face of cubemap.
    cudaGraphicsCubeFaceNegativeX
    Positive X face of cubemap.
    cudaGraphicsCubeFacePositiveY
    Positive Y face of cubemap.
    cudaGraphicsCubeFaceNegativeZ
    Positive Z face of cubemap.
    cudaGraphicsCubeFaceNegativeZ
    Negative Z face of cubemap.
    cudaGraphicsCubeFaceNegativeZ
    Negative Z face of cubemap.
```

4.20.2.6 enum cudaGraphicsMapFlags

CUDA graphics interop map flags

Enumerator:

```
    cudaGraphicsMapFlagsNone Default; Assume resource can be read/written.
    cudaGraphicsMapFlagsReadOnly CUDA will not write to this resource.
    cudaGraphicsMapFlagsWriteDiscard CUDA will only write to and will not read from this resource.
```

4.20.2.7 enum cudaGraphicsRegisterFlags

CUDA graphics interop register flags

Enumerator:

cudaGraphicsRegisterFlagsNone Default.

4.20.2.8 enum cudaMemcpyKind

CUDA memory copy types

Enumerator:

```
cudaMemcpyHostToHost Host -> Host.
cudaMemcpyHostToDevice Host -> Device.
cudaMemcpyDeviceToHost Device -> Host.
cudaMemcpyDeviceToDevice Device -> Device.
```

4.21 CUDA Driver API

Modules

- Initialization
- Device Management
- Version Management
- Context Management
- Module Management
- Stream Management
- Event Management
- Execution Control
- Memory Management
- Texture Reference Management
- OpenGL Interoperability
- Direct3D 9 Interoperability
- Direct3D 10 Interoperability
- Direct3D 11 Interoperability
- Graphics Interoperability
- Data types used by CUDA driver

4.21.1 Detailed Description

This section describes the low-level CUDA driver application programming interface.

4.22 Initialization 119

4.22 Initialization

Functions

• CUresult cuInit (unsigned int Flags)

Initialize the CUDA driver API.

4.22.1 Detailed Description

This section describes the initialization functions of the low-level CUDA driver application programming interface.

4.22.2 Function Documentation

4.22.2.1 cuInit (unsigned int Flags)

Initializes the driver API and must be called before any other function from the driver API. Currently, the Flags parameter must be 0. If cuInit() has not been called, any function from the driver API will return CUDA_ERROR_NOT_INITIALIZED.

Parameters:

Flags - Initialization flag for CUDA.

Returns:

CUDA_SUCCESS, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

4.23 Device Management

Functions

• CUresult cuDeviceComputeCapability (int *major, int *minor, CUdevice dev)

Returns the compute capability of the device.

• CUresult cuDeviceGet (CUdevice *device, int ordinal)

Returns a handle to a compute device.

• CUresult cuDeviceGetAttribute (int *pi, CUdevice_attribute attrib, CUdevice dev)

Returns information about the device.

CUresult cuDeviceGetCount (int *count)

Returns the number of compute-capable devices.

CUresult cuDeviceGetName (char *name, int len, CUdevice dev)

Returns an identifer string for the device.

• CUresult cuDeviceGetProperties (CUdevprop *prop, CUdevice dev)

Returns properties for a selected device.

CUresult cuDeviceTotalMem (unsigned int *bytes, CUdevice dev)

Returns the total amount of memory on the device.

4.23.1 Detailed Description

This section describes the device management functions of the low-level CUDA driver application programming interface.

4.23.2 Function Documentation

4.23.2.1 cuDeviceComputeCapability (int * major, int * minor, CUdevice dev)

Returns in *major and *minor the major and minor revision numbers that define the compute capability of the device dev.

Parameters:

```
major - Major revision numberminor - Minor revision numberdev - Device handle
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceGetAttribute, cuDeviceGetCount, cuDeviceGetName, cuDeviceGet, cuDeviceGetProperties, cuDevice-TotalMem

4.23.2.2 cuDeviceGet (CUdevice * device, int ordinal)

Returns in *device a device handle given an ordinal in the range [0, cuDeviceGetCount()-1].

Parameters:

```
device - Returned device handleordinal - Device number to get handle for
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceComputeCapability, cuDeviceGetAttribute, cuDeviceGetCount, cuDeviceGetName, cuDeviceGetProperties, cuDeviceTotalMem

4.23.2.3 cuDeviceGetAttribute (int * pi, CUdevice_attribute attrib, CUdevice dev)

Returns in *pi the integer value of the attribute attrib on device dev. The supported attributes are:

- CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK: Maximum number of threads per block;
- CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X: Maximum x-dimension of a block;
- CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y: Maximum y-dimension of a block;
- CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z: Maximum z-dimension of a block;
- CU DEVICE ATTRIBUTE MAX GRID DIM X: Maximum x-dimension of a grid;
- CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y: Maximum y-dimension of a grid;
- CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z: Maximum z-dimension of a grid;
- CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK: Maximum amount of shared memory available to a thread block in bytes; this amount is shared by all thread blocks simultaneously resident on a multiprocessor;
- CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY: Memory available on device for __constant_variables in a CUDA C kernel in bytes;
- CU_DEVICE_ATTRIBUTE_WARP_SIZE: Warp size in threads;
- CU_DEVICE_ATTRIBUTE_MAX_PITCH: Maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through cuMemAllocPitch();

• CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK: Maximum number of 32-bit registers available to a thread block; this number is shared by all thread blocks simultaneously resident on a multiprocessor;

- CU_DEVICE_ATTRIBUTE_CLOCK_RATE: Peak clock frequency in kilohertz;
- CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT: Alignment requirement; texture base addresses aligned to textureAlign bytes do not need an offset applied to texture fetches;
- CU_DEVICE_ATTRIBUTE_GPU_OVERLAP: 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not;
- CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT: Number of multiprocessors on the device;
- CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT: 1 if there is a run time limit for kernels executed on the device, or 0 if not;
- CU_DEVICE_ATTRIBUTE_INTEGRATED: 1 if the device is integrated with the memory subsystem, or 0 if not:
- CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY: 1 if the device can map host memory into the CUDA address space, or 0 if not;
- CU_DEVICE_ATTRIBUTE_COMPUTE_MODE: Compute mode that device is currently in. Available modes
 are as follows:
 - CU_COMPUTEMODE_DEFAULT: Default mode Device is not restricted and can have multiple CUDA
 contexts present at a single time.
 - CU_COMPUTEMODE_EXCLUSIVE: Compute-exclusive mode Device can have only one CUDA context present on it at a time.
 - CU_COMPUTEMODE_PROHIBITED: Compute-prohibited mode Device is prohibited from creating new CUDA contexts.
- CU_DEVICE_ATTRIBUTE_CONCURRENT_KERNELS: 1 if the device supports executing multiple kernels within the same context simultaneously, or 0 if not. It is not guaranteed that multiple kernels will be resident on the device concurrently so this feature should not be relied upon for correctness;
- CU_DEVICE_ATTRIBUTE_ECC_ENABLED: 1 if error correction is enabled on the device, 0 if error correction is disabled or not supported by the device.

Parameters:

pi - Returned device attribute valueattrib - Device attribute to query

dev - Device handle

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceComputeCapability, cuDeviceGetCount, cuDeviceGetName, cuDeviceGet, cuDeviceGetProperties, cuDeviceTotalMem

4.23.2.4 cuDeviceGetCount (int * count)

Returns in *count the number of devices with compute capability greater than or equal to 1.0 that are available for execution. If there is no such device, cuDeviceGetCount() returns 0.

Parameters:

count - Returned number of compute-capable devices

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceComputeCapability, cuDeviceGetAttribute, cuDeviceGetName, cuDeviceGet, cuDeviceGetProperties, cuDeviceTotalMem

4.23.2.5 cuDeviceGetName (char * name, int len, CUdevice dev)

Returns an ASCII string identifying the device dev in the NULL-terminated string pointed to by name. len specifies the maximum length of the string that may be returned.

Parameters:

name - Returned identifier string for the device

len - Maximum length of string to store in name

dev - Device to get identifier string for

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceComputeCapability, cuDeviceGetAttribute, cuDeviceGetCount, cuDeviceGet, cuDeviceGetProperties, cuDeviceTotalMem

4.23.2.6 cuDeviceGetProperties (CUdevprop * prop, CUdevice dev)

Returns in *prop the properties of device dev. The CUdevprop structure is defined as:

```
typedef struct CUdevprop_st {
  int maxThreadsPerBlock;
  int maxThreadsDim[3];
  int maxGridSize[3];
  int sharedMemPerBlock;
  int totalConstantMemory;
  int SIMDWidth;
  int memPitch;
  int regsPerBlock;
  int clockRate;
  int textureAlign
} CUdevprop;
```

where:

- maxThreadsPerBlock is the maximum number of threads per block;
- maxThreadsDim[3] is the maximum sizes of each dimension of a block;
- maxGridSize[3] is the maximum sizes of each dimension of a grid;
- sharedMemPerBlock is the total amount of shared memory available per block in bytes;
- totalConstantMemory is the total amount of constant memory available on the device in bytes;
- SIMDWidth is the warp size;
- memPitch is the maximum pitch allowed by the memory copy functions that involve memory regions allocated through cuMemAllocPitch();
- regsPerBlock is the total number of registers available per block;
- clockRate is the clock frequency in kilohertz;
- textureAlign is the alignment requirement; texture base addresses that are aligned to textureAlign bytes do not need an offset applied to texture fetches.

Parameters:

```
prop - Returned properties of devicedev - Device to get properties for
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceComputeCapability, cuDeviceGetAttribute, cuDeviceGetCount, cuDeviceGetName, cuDeviceGet, cuDeviceTotalMem

4.23.2.7 cuDeviceTotalMem (unsigned int * bytes, CUdevice dev)

Returns in *bytes the total amount of memory available on the device dev in bytes.

Parameters:

```
bytes - Returned memory available on device in bytesdev - Device handle
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_DEVICE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceGetOunt, cuDeviceGetName, cuDeviceGetCount, cuDeviceGetName, cuDeviceGet, cuDeviceGetProperties,

4.24 Version Management

Functions

• CUresult cuDriverGetVersion (int *driverVersion)

Returns the CUDA driver version.

4.24.1 Detailed Description

This section describes the version management functions of the low-level CUDA driver application programming interface.

4.24.2 Function Documentation

4.24.2.1 cuDriverGetVersion (int * driverVersion)

Returns in *driverVersion the version number of the installed CUDA driver. This function automatically returns CUDA_ERROR_INVALUE if the driverVersion argument is NULL.

Parameters:

driver Version - Returns the CUDA driver version

Returns:

CUDA_SUCCESS, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

4.25 Context Management

Functions

• CUresult cuCtxAttach (CUcontext *pctx, unsigned int flags)

Increment a context's usage-count.

• CUresult cuCtxCreate (CUcontext *pctx, unsigned int flags, CUdevice dev)

Create a CUDA context.

• CUresult cuCtxDestroy (CUcontext ctx)

Destroy the current context or a floating CUDA context.

• CUresult cuCtxDetach (CUcontext ctx)

Decrement a context's usage-count.

• CUresult cuCtxGetDevice (CUdevice *device)

Returns the device ID for the current context.

CUresult cuCtxPopCurrent (CUcontext *pctx)

Pops the current CUDA context from the current CPU thread.

• CUresult cuCtxPushCurrent (CUcontext ctx)

Pushes a floating context on the current CPU thread.

• CUresult cuCtxSynchronize (void)

Block for a context's tasks to complete.

4.25.1 Detailed Description

This section describes the context management functions of the low-level CUDA driver application programming interface.

4.25.2 Function Documentation

4.25.2.1 cuCtxAttach (CUcontext * pctx, unsigned int flags)

Increments the usage count of the context and passes back a context handle in *pctx that must be passed to cuCtxDetach() when the application is done with the context. cuCtxAttach() fails if there is no context current to the thread.

Currently, the flags parameter must be 0.

Parameters:

pctx - Returned context handle of the current context

flags - Context attach flags (must be 0)

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxCreate, cuCtxDestroy, cuCtxDetach, cuCtxGetDevice, cuCtxPopCurrent, cuCtxPushCurrent, cuCtxSynchronize

4.25.2.2 cuCtxCreate (CUcontext * pctx, unsigned int flags, CUdevice dev)

Creates a new CUDA context and associates it with the calling thread. The flags parameter is described below. The context is created with a usage count of 1 and the caller of cuCtxCreate() must call cuCtxDestroy() or cuCtxDetach() when done using the context. If a context is already current to the thread, it is supplanted by the newly created context and may be restored by a subsequent call to cuCtxPopCurrent().

The two LSBs of the flags parameter can be used to control how the OS thread, which owns the CUDA context at the time of an API call, interacts with the OS scheduler when waiting for results from the GPU.

- CU_CTX_SCHED_AUTO: The default value if the flags parameter is zero, uses a heuristic based on the number of active CUDA contexts in the process *C* and the number of logical processors in the system *P*. If *C* > *P*, then CUDA will yield to other OS threads when waiting for the GPU, otherwise CUDA will not yield while waiting for results and actively spin on the processor.
- CU_CTX_SCHED_SPIN: Instruct CUDA to actively spin when waiting for results from the GPU. This can decrease latency when waiting for the GPU, but may lower the performance of CPU threads if they are performing work in parallel with the CUDA thread.
- CU_CTX_SCHED_YIELD: Instruct CUDA to yield its thread when waiting for results from the GPU. This can increase latency when waiting for the GPU, but can increase the performance of CPU threads performing work in parallel with the GPU.
- CU_CTX_BLOCKING_SYNC: Instruct CUDA to block the CPU thread on a synchronization primitive when waiting for the GPU to finish work.
- CU_CTX_MAP_HOST: Instruct CUDA to support mapped pinned allocations. This flag must be set in order to allocate pinned host memory that is accessible to the GPU.
- CU_CTX_LMEM_RESIZE_TO_MAX: Instruct CUDA to not reduce local memory after resizing local memory
 for a kernel. This can prevent thrashing by local memory allocations when launching many kernels with high
 local memory usage at the cost of potentially increased memory usage.

Note to Linux users:

Context creation will fail with CUDA_ERROR_UNKNOWN if the compute mode of the device is CU_COMPUTEMODE_PROHIBITED. Similarly, context creation will also fail with CUDA_ERROR_UNKNOWN if the compute mode for the device is set to CU_COMPUTEMODE_EXCLUSIVE and there is already an active context on the device. The function cuDeviceGetAttribute() can be used with CU_DEVICE_ATTRIBUTE_COMPUTE_MODE to determine the compute mode of the device. The *nvidia-smi* tool can be used to set the compute mode for devices. Documentation for *nvidia-smi* can be obtained by passing a -h option to it.

Parameters:

pctx - Returned context handle of the new context

flags - Context creation flags

dev - Device to create context on

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_DEVICE, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuCtxAttach,\ cuCtxDestroy,\ cuCtxDetach,\ cuCtxGetDevice,\ cuCtxPopCurrent,\ cuCtxPushCurrent,\ cuCtxSynchronize$

4.25.2.3 cuCtxDestroy (CUcontext ctx)

Destroys the CUDA context specified by ctx. If the context usage count is not equal to 1, or the context is current to any CPU thread other than the current one, this function fails. Floating contexts (detached from a CPU thread via cuCtxPopCurrent()) may be destroyed by this function.

Parameters:

ctx - Context to destroy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxAttach, cuCtxCreate, cuCtxDetach, cuCtxGetDevice, cuCtxPopCurrent, cuCtxPushCurrent, cuCtxSynchronize

4.25.2.4 cuCtxDetach (CUcontext ctx)

Decrements the usage count of the context ctx, and destroys the context if the usage count goes to 0. The context must be a handle that was passed back by cuCtxCreate() or cuCtxAttach(), and must be current to the calling thread.

Parameters:

ctx - Context to destroy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxAttach, cuCtxCreate, cuCtxDestroy, cuCtxGetDevice, cuCtxPopCurrent, cuCtxPushCurrent, cuCtxSynchronize

4.25.2.5 cuCtxGetDevice (CUdevice * device)

Returns in *device the ordinal of the current context's device.

Parameters:

device - Returned device ID for the current context

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxAttach, cuCtxCreate, cuCtxDestroy, cuCtxDetach, cuCtxPopCurrent, cuCtxPushCurrent, cuCtxSynchronize

4.25.2.6 cuCtxPopCurrent (CUcontext * pctx)

Pops the current CUDA context from the CPU thread. The CUDA context must have a usage count of 1. CUDA contexts have a usage count of 1 upon creation; the usage count may be incremented with cuCtxAttach() and decremented with cuCtxDetach().

If successful, cuCtxPopCurrent() passes back the old context handle in *pctx. That context may then be made current to a different CPU thread by calling cuCtxPushCurrent().

Floating contexts may be destroyed by calling cuCtxDestroy().

If a context was current to the CPU thread before cuCtxCreate() or cuCtxPushCurrent() was called, this function makes that context current to the CPU thread again.

Parameters:

pctx - Returned new context handle

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxAttach, cuCtxCreate, cuCtxDestroy, cuCtxDetach, cuCtxGetDevice, cuCtxPushCurrent, cuCtxSynchronize

4.25.2.7 cuCtxPushCurrent (CUcontext ctx)

Pushes the given context ctx onto the CPU thread's stack of current contexts. The specified context becomes the CPU thread's current context, so all CUDA functions that operate on the current context are affected.

The previous current context may be made current again by calling cuCtxDestroy() or cuCtxPopCurrent().

The context must be "floating," i.e. not attached to any thread. Contexts are made to float by calling cuCtxPopCurrent().

Parameters:

ctx - Floating context to attach

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxAttach, cuCtxCreate, cuCtxDestroy, cuCtxDetach, cuCtxGetDevice, cuCtxPopCurrent, cuCtxSynchronize

4.25.2.8 cuCtxSynchronize (void)

Blocks until the device has completed all preceding requested tasks. cuCtxSynchronize() returns an error if one of the preceding tasks failed. If the context was created with the CU_CTX_BLOCKING_SYNC flag, the CPU thread will block until the GPU context has finished its work.

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxAttach, cuCtxCreate, cuCtxDestroy, cuCtxDetach, cuCtxGetDevice, cuCtxPopCurrent, cuCtxSynchronize

4.26 Module Management

Functions

• CUresult cuModuleGetFunction (CUfunction *hfunc, CUmodule hmod, const char *name)

Returns a function handle.

- CUresult cuModuleGetGlobal (CUdeviceptr *dptr, unsigned int *bytes, CUmodule hmod, const char *name)

 Returns a global pointer from a module.
- CUresult cuModuleGetTexRef (CUtexref *pTexRef, CUmodule hmod, const char *name)

 Returns a handle to a texture-reference.
- CUresult cuModuleLoad (CUmodule *module, const char *fname)

 Loads a compute module.
- CUresult cuModuleLoadData (CUmodule *module, const void *image)

 Load a module's data.
- CUresult cuModuleLoadDataEx (CUmodule *module, const void *image, unsigned int numOptions, CUjit_option *options, void **optionValues)

Load a module's data with options.

- CUresult cuModuleLoadFatBinary (CUmodule *module, const void *fatCubin)
 Load a module's data.
- CUresult cuModuleUnload (CUmodule hmod)

4.26.1 Detailed Description

Unloads a module.

This section describes the module management functions of the low-level CUDA driver application programming interface.

4.26.2 Function Documentation

4.26.2.1 cuModuleGetFunction (CUfunction * hfunc, CUmodule hmod, const char * name)

Returns in *hfunc the handle of the function of name name located in module hmod. If no function of that name exists, cuModuleGetFunction() returns CUDA_ERROR_NOT_FOUND.

Parameters:

hfunc - Returned function handle

hmod - Module to retrieve function from

name - Name of function to retrieve

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE, CUDA ERROR NOT FOUND

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuModuleGetGlobal,\ cuModuleGetTexRef,\ cuModuleLoad,\ cuModuleLoadData,\ cuModuleLoadDataEx,\ cuModuleLoadFatBinary,\ cuModuleUnload$

4.26.2.2 cuModuleGetGlobal (CUdeviceptr * dptr, unsigned int * bytes, CUmodule hmod, const char * name)

Returns in *dptr and *bytes the base pointer and size of the global of name name located in module hmod. If no variable of that name exists, cuModuleGetGlobal() returns CUDA_ERROR_NOT_FOUND. Both parameters dptr and bytes are optional. If one of them is NULL, it is ignored.

Parameters:

dptr - Returned global device pointer

bytes - Returned global size in bytes

hmod - Module to retrieve function from

name - Name of global to retrieve

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE, CUDA ERROR NOT FOUND

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuModuleGetFunction, cuModuleGetTexRef, cuModuleLoad, cuModuleLoadData, cuModuleLoadDataEx, cuModuleLoadFatBinary, cuModuleUnload

4.26.2.3 cuModuleGetTexRef (CUtexref * pTexRef, CUmodule hmod, const char * name)

Returns in *pTexRef the handle of the texture reference of name name in the module hmod. If no texture reference of that name exists, cuModuleGetTexRef() returns CUDA_ERROR_NOT_FOUND. This texture reference handle should not be destroyed, since it will be destroyed when the module is unloaded.

Parameters:

pTexRef - Returned global device pointer

hmod - Module to retrieve texture-reference from

name - Name of texture-reference to retrieve

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_NOT_FOUND

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuModuleGetFunction, cuModuleGetGlobal, cuModuleLoad, cuModuleLoadData, cuModuleLoadDataEx, cuModuleLoadFatBinary, cuModuleUnload

4.26.2.4 cuModuleLoad (CUmodule * module, const char * fname)

Takes a filename fname and loads the corresponding module module into the current context. The CUDA driver API does not attempt to lazily allocate the resources needed by a module; if the memory for functions and data (constant and global) needed by the module cannot be allocated, cuModuleLoad() fails. The file should be a *cubin* file as output by **nvcc** or a *PTX* file, either as output by **nvcc** or handwrtten.

Parameters:

```
module - Returned modulefname - Filename of module to load
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_NOT_FOUND, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_FILE_NOT_FOUND

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuModuleGetFunction, cuModuleGetGlobal, cuModuleGetTexRef, cuModuleLoadData, cuModuleLoadDataEx, cuModuleLoadFatBinary, cuModuleUnload

4.26.2.5 cuModuleLoadData (CUmodule * module, const void * image)

Takes a pointer image and loads the corresponding module module into the current context. The pointer may be obtained by mapping a *cubin* or *PTX* file, passing a *cubin* or *PTX* file as a NULL-terminated text string, or incorporating a *cubin* object into the executable resources and using operating system calls such as Windows FindResource () to obtain the pointer.

Parameters:

```
module - Returned moduleimage - Module data to load
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuModuleGetFunction, cuModuleGetGlobal, cuModuleGetTexRef, cuModuleLoad, cuModuleLoadDataEx, cu-ModuleLoadFatBinary, cuModuleUnload

4.26.2.6 cuModuleLoadDataEx (CUmodule * module, const void * image, unsigned int numOptions, CUjit_option * options, void ** optionValues)

Takes a pointer image and loads the corresponding module module into the current context. The pointer may be obtained by mapping a *cubin* or *PTX* file, passing a *cubin* or *PTX* file as a NULL-terminated text string, or incorporating a *cubin* object into the executable resources and using operating system calls such as Windows FindResource () to obtain the pointer. Options are passed as an array via options and any corresponding parameters are passed in optionValues. The number of total options is supplied via numOptions. Any outputs will be returned via optionValues. Supported options are (types for the option values are specified in parentheses after the option name):

- CU_JIT_MAX_REGISTERS: (unsigned int) input specifies the maximum number of registers per thread;
- CU_JIT_THREADS_PER_BLOCK: (unsigned int) input specifies number of threads per block to target compilation for; output returns the number of threads the compiler actually targeted;
- CU_JIT_WALL_TIME: (float) output returns the float value of wall clock time, in milliseconds, spent compiling the *PTX* code;
- CU_JIT_INFO_LOG_BUFFER: (char*) input is a pointer to a buffer in which to print any informational log messages from *PTX* assembly (the buffer size is specified via option CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES);
- CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES: (unsigned int) input is the size in bytes of the buffer; output is the number of bytes filled with messages;
- CU_JIT_ERROR_LOG_BUFFER: (char*) input is a pointer to a buffer in which to print any error log messages from PTX assembly (the buffer size is specified via option CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES);
- CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES: (unsigned int) input is the size in bytes of the buffer; output is the number of bytes filled with messages;
- CU_JIT_OPTIMIZATION_LEVEL: (unsigned int) input is the level of optimization to apply to generated code (0 4), with 4 being the default and highest level;
- CU_JIT_TARGET_FROM_CUCONTEXT: (No option value) causes compilation target to be determined based on current attached context (default);
- CU_JIT_TARGET: (unsigned int for enumerated type CUjit_target_enum) input is the compilation target based on supplied CUjit_target_enum; possible values are:
 - CU TARGET COMPUTE 10
 - CU TARGET COMPUTE 11
 - CU_TARGET_COMPUTE_12
 - CU_TARGET_COMPUTE_13
 - CU_TARGET_COMPUTE_20
- CU_JIT_FALLBACK_STRATEGY: (unsigned int for enumerated type CUjit_fallback_enum) chooses fallback strategy if matching cubin is not found; possible values are:
 - CU_PREFER_PTX

- CU PREFER BINARY

Parameters:

```
    module - Returned module
    image - Module data to load
    numOptions - Number of options
    options - Options for JIT
    optionValues - Option values for JIT
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_NO_BINARY_FOR_GPU

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuModuleGetFunction, cuModuleGetGlobal, cuModuleGetTexRef, cuModuleLoad, cuModuleLoadData, cuModuleLoadFatBinary, cuModuleUnload

4.26.2.7 cuModuleLoadFatBinary (CUmodule * module, const void * fatCubin)

Takes a pointer fatCubin and loads the corresponding module module into the current context. The pointer represents a *fat binary* object, which is a collection of different *cubin* files, all representing the same device code, but compiled and optimized for different architectures. There is currently no documented API for constructing and using fat binary objects by programmers, and therefore this function is an internal function in this version of CUDA. More information can be found in the **nvcc** document.

Parameters:

```
module - Returned modulefatCubin - Fat binary to load
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_NOT_FOUND, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_NO_BINARY_FOR_GPU

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuModuleGetFunction,\ cuModuleGetGlobal,\ cuModuleGetTexRef,\ cuModuleLoad,\ cuModuleLoadData,\ cuModuleLoadDataEx,\ cuModuleUnload$

4.26.2.8 cuModuleUnload (CUmodule hmod)

Unloads a module hmod from the current context.

Parameters:

hmod - Module to unload

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuModuleGetFunction,\ cuModuleGetGlobal,\ cuModuleGetTexRef,\ cuModuleLoad,\ cuModuleLoadData,\ cuModuleLoadDataEx,\ cuModuleLoadFatBinary$

4.27 Stream Management

Functions

• CUresult cuStreamCreate (CUstream *phStream, unsigned int Flags)

Create a stream.

• CUresult cuStreamDestroy (CUstream hStream)

Destroys a stream.

• CUresult cuStreamQuery (CUstream hStream)

Determine status of a compute stream.

• CUresult cuStreamSynchronize (CUstream hStream)

Wait until a stream's tasks are completed.

4.27.1 Detailed Description

This section describes the stream management functions of the low-level CUDA driver application programming interface.

4.27.2 Function Documentation

4.27.2.1 cuStreamCreate (CUstream * phStream, unsigned int Flags)

Creates a stream and returns a handle in phStream. Flags is required to be 0.

Parameters:

```
phStream - Returned newly created stream
```

 ${\it Flags}\,$ - Parameters for stream creation (must be 0)

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuStreamDestroy, cuStreamQuery, cuStreamSynchronize

4.27.2.2 cuStreamDestroy (CUstream hStream)

Destroys the stream specified by hStream.

Parameters:

hStream - Stream to destroy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuStreamCreate, cuStreamQuery, cuStreamSynchronize

4.27.2.3 cuStreamQuery (CUstream hStream)

Returns CUDA_SUCCESS if all operations in the stream specified by hStream have completed, or CUDA_ERROR_NOT_READY if not.

Parameters:

hStream - Stream to query status of

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_READY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuStreamCreate, cuStreamDestroy, cuStreamSynchronize

4.27.2.4 cuStreamSynchronize (CUstream hStream)

Waits until the device has completed all operations in the stream specified by hStream. If the context was created with the CU_CTX_BLOCKING_SYNC flag, the CPU thread will block until the stream is finished with all of its tasks.

Parameters:

hStream - Stream to wait for

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuStreamCreate, cuStreamDestroy, cuStreamQuery

4.28 Event Management

Functions

• CUresult cuEventCreate (CUevent *phEvent, unsigned int Flags)

Creates an event.

• CUresult cuEventDestroy (CUevent hEvent)

Destroys an event.

CUresult cuEventElapsedTime (float *pMilliseconds, CUevent hStart, CUevent hEnd)

Computes the elapsed time between two events.

• CUresult cuEventQuery (CUevent hEvent)

Queries an event's status.

• CUresult cuEventRecord (CUevent hEvent, CUstream hStream)

Records an event.

• CUresult cuEventSynchronize (CUevent hEvent)

Waits for an event to complete.

4.28.1 Detailed Description

This section describes the event management functions of the low-level CUDA driver application programming interface.

4.28.2 Function Documentation

4.28.2.1 cuEventCreate (CUevent * phEvent, unsigned int Flags)

Creates an event *phEvent with the flags specified via Flags. Valid flags include:

- CU_EVENT_DEFAULT: Default event creation flag.
- CU_EVENT_BLOCKING_SYNC: Specifies that event should use blocking synchronization. A CPU thread
 that uses cuEventSynchronize() to wait on an event created with this flag will block until the event has actually
 been recorded.

Parameters:

```
phEvent - Returns newly created eventFlags - Event creation flags
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuEventRecord, cuEventQuery, cuEventSynchronize, cuEventDestroy, cuEventElapsedTime

4.28.2.2 cuEventDestroy (CUevent hEvent)

Destroys the event specified by event.

Parameters:

hEvent - Event to destroy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuEventCreate, cuEventRecord, cuEventQuery, cuEventSynchronize, cuEventElapsedTime

4.28.2.3 cuEventElapsedTime (float * pMilliseconds, CUevent hStart, CUevent hEnd)

Computes the elapsed time between two events (in milliseconds with a resolution of around 0.5 microseconds). If either event has not been recorded yet, this function returns CUDA_ERROR_NOT_READY. If either event has been recorded with a non-zero stream, the result is undefined.

Parameters:

```
pMilliseconds - Returned elapsed time in millisecondshStart - Starting eventhEnd - Ending event
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_READY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuEventCreate, cuEventRecord, cuEventQuery, cuEventSynchronize, cuEventDestroy

4.28.2.4 cuEventQuery (CUevent hEvent)

Returns CUDA_SUCCESS if the event has actually been recorded, or CUDA_ERROR_NOT_READY if not. If cuEventRecord() has not been called on this event, the function returns CUDA_ERROR_INVALID_VALUE.

Parameters:

hEvent - Event to query

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_NOT_READY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuEventCreate, cuEventRecord, cuEventSynchronize, cuEventDestroy, cuEventElapsedTime

4.28.2.5 cuEventRecord (CUevent hEvent, CUstream hStream)

Records an event. If stream is non-zero, the event is recorded after all preceding operations in the stream have been completed; otherwise, it is recorded after all preceding operations in the CUDA context have been completed. Since operation is asynchronous, cuEventQuery() and/or cuEventSynchronize() must be used to determine when the event has actually been recorded.

If cuEventRecord() has previously been called and the event has not been recorded yet, this function returns CUDA_-ERROR_INVALID_VALUE.

Parameters:

hEvent - Event to record

hStream - Stream to record event for

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuEventCreate, cuEventQuery, cuEventSynchronize, cuEventDestroy, cuEventElapsedTime

4.28.2.6 cuEventSynchronize (CUevent *hEvent*)

Waits until the event has actually been recorded. If cuEventRecord() has been called on this event, the function returns CUDA_ERROR_INVALID_VALUE. Waiting for an event that was created with the CU_EVENT_BLOCKING_-SYNC flag will cause the calling CPU thread to block until the event has actually been recorded.

If cuEventRecord() has previously been called and the event has not been recorded yet, this function returns CUDA_ERROR_INVALID_VALUE.

Parameters:

hEvent - Event to wait for

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuEventCreate, cuEventRecord, cuEventQuery, cuEventDestroy, cuEventElapsedTime

4.29 Execution Control

Functions

• CUresult cuFuncGetAttribute (int *pi, CUfunction_attribute attrib, CUfunction hfunc)

Returns information about a function.

• CUresult cuFuncSetBlockShape (CUfunction hfunc, int x, int y, int z)

Sets the block-dimensions for the function.

• CUresult cuFuncSetCacheConfig (CUfunction hfunc, CUfunc_cache config)

Sets the preferred cache configuration for a device function.

• CUresult cuFuncSetSharedSize (CUfunction hfunc, unsigned int bytes)

Sets the dynamic shared-memory size for the function.

• CUresult cuLaunch (CUfunction f)

Launches a CUDA function.

• CUresult cuLaunchGrid (CUfunction f, int grid_width, int grid_height)

Launches a CUDA function.

CUresult cuLaunchGridAsync (CUfunction f, int grid_width, int grid_height, CUstream hStream)

Launches a CUDA function.

• CUresult cuParamSetf (CUfunction hfunc, int offset, float value)

Adds a floating-point parameter to the function's argument list.

• CUresult cuParamSeti (CUfunction hfunc, int offset, unsigned int value)

Adds an integer parameter to the function's argument list.

• CUresult cuParamSetSize (CUfunction hfunc, unsigned int numbytes)

Sets the parameter size for the function.

• CUresult cuParamSetTexRef (CUfunction hfunc, int texunit, CUtexref hTexRef)

Adds a texture-reference to the function's argument list.

CUresult cuParamSety (CUfunction hfunc, int offset, void *ptr, unsigned int numbytes)

Adds arbitrary data to the function's argument list.

4.29.1 Detailed Description

This section describes the execution control functions of the low-level CUDA driver application programming interface.

4.29 Execution Control 145

4.29.2 Function Documentation

4.29.2.1 cuFuncGetAttribute (int * pi, CUfunction_attribute attrib, CUfunction hfunc)

Returns in *pi the integer value of the attribute attrib on the kernel given by hfunc. The supported attributes are:

- CU_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK: The number of threads beyond which a launch
 of the function would fail. This number depends on both the function and the device on which the function is
 currently loaded.
- CU_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES: The size in bytes of statically-allocated shared memory required by this function. This does not include dynamically-allocated shared memory requested by the user at runtime.
- CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES: The size in bytes of user-allocated constant memory required by this function.
- CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES: The size in bytes of thread local memory used by this function.
- CU_FUNC_ATTRIBUTE_NUM_REGS: The number of registers used by each thread of this function.
- CU_FUNC_ATTRIBUTE_PTX_VERSION: The PTX virtual architecture version for which the function was compiled. This value is the major PTX version * 10 + the minor PTX version, so a PTX version 1.3 function would return the value 13. Note that this may return the undefined value of 0 for cubins compiled prior to CUDA 3.0.
- CU_FUNC_ATTRIBUTE_BINARY_VERSION: The binary version for which the function was compiled. This value is the major binary version * 10 + the minor binary version, so a binary version 1.3 function would return the value 13. Note that this will return a value of 10 for legacy cubins that do not have a properly-encoded binary architecture version.

Parameters:

pi - Returned attribute value

attrib - Attribute requested

hfunc - Function to query attribute of

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncSetCacheConfig, cuParamSetSize, cuParamSeti, cuParamSetf, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29.2.2 cuFuncSetBlockShape (CUfunction hfunc, int x, int y, int z)

Specifies the x, y, and z dimensions of the thread blocks that are created when the kernel given by hfunc is launched.

Parameters:

hfunc - Kernel to specify dimensions of

x - X dimension

y - Y dimension

z - Z dimension

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetSharedSize, cuFuncSetCacheConfig, cuFuncGetAttribute, cuParamSetSize, cuParamSeti, cuParamSetf, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29.2.3 cuFuncSetCacheConfig (CUfunction *hfunc*, CUfunc_cache *config*)

On devices where the L1 cache and shared memory use the same hardware resources, this sets through config the preferred cache configuration for the device function hfunc. This is only a preference. The driver will use the requested configuration if possible, but it is free to choose a different configuration if required to execute hfunc.

This setting does nothing on devices where the size of the L1 cache and shared memory are fixed.

Switching between configuration modes may insert a device-side synchronization point for streamed kernel launches.

The supported cache modes are:

- CU_FUNC_CACHE_PREFER_NONE: no preference for shared memory or L1 (default)
- CU_FUNC_CACHE_PREFER_SHARED: function prefers larger shared memory and smaller L1 cache.
- CU_FUNC_CACHE_PREFER_L1: function prefers larger L1 cache and smaller shared memory.

Parameters:

```
hfunc - Kernel to configure cache forconfig - Requested cache configuration
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncGetAttribute, cuParamSetSize, cuParamSeti, cuParamSetf, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29 Execution Control 147

4.29.2.4 cuFuncSetSharedSize (CUfunction *hfunc*, unsigned int *bytes*)

Sets through bytes the amount of dynamic shared memory that will be available to each thread block when the kernel given by hfunc is launched.

Parameters:

hfunc - Kernel to specify dynamic shared-memory size for

bytes - Dynamic shared-memory size per thread in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID HANDLE, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetCacheConfig, cuFuncGetAttribute, cuParamSetSize, cuParamSeti, cuParamSetf, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29.2.5 cuLaunch (CUfunction f)

Invokes the kernel f on a 1 x 1 x 1 grid of blocks. The block contains the number of threads specified by a previous call to cuFuncSetBlockShape().

Parameters:

f - Kernel to launch

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_LAUNCH_FAILED, CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES, CUDA_ERROR_LAUNCH_TIMEOUT, CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSetf, cuParamSeti, cuParamSetv, cuParamSetTexRef, cuLaunchGrid, cuLaunchGridAsync

4.29.2.6 cuLaunchGrid (CUfunction f, int grid_width, int grid_height)

Invokes the kernel f on a grid_width x grid_height grid of blocks. Each block contains the number of threads specified by a previous call to cuFuncSetBlockShape().

Parameters:

```
f - Kernel to launchgrid_width - Width of grid in blocksgrid_height - Height of grid in blocks
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_LAUNCH_FAILED, CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES, CUDA_ERROR_LAUNCH_TIMEOUT, CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSetf, cuParamSeti, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGridAsync

4.29.2.7 cuLaunchGridAsync (CUfunction f, int grid_width, int grid_height, CUstream hStream)

Invokes the kernel f on a grid_width x grid_height grid of blocks. Each block contains the number of threads specified by a previous call to cuFuncSetBlockShape().

cuLaunchGridAsync() can optionally be associated to a stream by passing a non-zero hStream argument.

Parameters:

```
    f - Kernel to launch
    grid_width - Width of grid in blocks
    grid_height - Height of grid in blocks
    hStream - Stream identifier
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_LAUNCH_FAILED, CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES, CUDA_ERROR_LAUNCH_TIMEOUT, CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSetf, cuParamSeti, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid

4.29 Execution Control

4.29.2.8 cuParamSetf (CUfunction hfunc, int offset, float value)

Sets a floating-point parameter that will be specified the next time the kernel corresponding to hfunc will be invoked. offset is a byte offset.

Parameters:

```
hfunc - Kernel to add parameter to
```

offset - Offset to add parameter to argument list

value - Value of parameter

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSeti, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29.2.9 cuParamSeti (CUfunction hfunc, int offset, unsigned int value)

Sets an integer parameter that will be specified the next time the kernel corresponding to hfunc will be invoked. offset is a byte offset.

Parameters:

```
hfunc - Kernel to add parameter to
```

offset - Offset to add parameter to argument list

value - Value of parameter

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSetf, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29.2.10 cuParamSetSize (CUfunction hfunc, unsigned int numbytes)

Sets through numbytes the total size in bytes needed by the function parameters of the kernel corresponding to hfunc.

Parameters:

```
hfunc - Kernel to set parameter size fornumbytes - Size of parameter list in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetf, cuParamSeti, cuParamSetv, cuParamSetTexRef, cuLaunch, cuLaunchGrid, cuLaunchGridAsync

4.29.2.11 cuParamSetTexRef (CUfunction hfunc, int texunit, CUtexref hTexRef)

Makes the CUDA array or linear memory bound to the texture reference hTexRef available to a device program as a texture. In this version of CUDA, the texture-reference must be obtained via cuModuleGetTexRef() and the texturit parameter must be set to CU_PARAM_TR_DEFAULT.

Parameters:

```
hfunc - Kernel to add texture-reference totexunit - Texture unit (must be CU_PARAM_TR_DEFAULT)hTexRef - Texture-reference to add to argument list
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSetf, cuParamSeti, cuParamSetv, cuLaunchGrid, cuLaunchGridAsync

4.29.2.12 cuParamSetv (CUfunction hfunc, int offset, void * ptr, unsigned int numbytes)

Copies an arbitrary amount of data (specified in numbytes) from ptr into the parameter space of the kernel corresponding to hfunc. offset is a byte offset.

4.29 Execution Control 151

Parameters:

```
hfunc - Kernel to add data to
offset - Offset to add data to argument list
ptr - Pointer to arbitrary data
numbytes - Size of data to copy in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuFuncSetBlockShape, cuFuncSetSharedSize, cuFuncGetAttribute, cuParamSetSize, cuParamSetf, cuParamSeti, cuParamSetTexRef, cuLaunchGrid, cuLaunchGridAsync

4.30 Memory Management

Functions

• CUresult cuArray3DCreate (CUarray *pHandle, const CUDA_ARRAY3D_DESCRIPTOR *pAllocateArray)

Creates a 3D CUDA array.

CUresult cuArray3DGetDescriptor (CUDA_ARRAY3D_DESCRIPTOR *pArrayDescriptor, CUarray hArray)

Get a 3D CUDA array descriptor.

- CUresult cuArrayCreate (CUarray *pHandle, const CUDA_ARRAY_DESCRIPTOR *pAllocateArray)
 Creates a 1D or 2D CUDA array.
- CUresult cuArrayDestroy (CUarray hArray)

Destroys a CUDA array.

- CUresult cuArrayGetDescriptor (CUDA_ARRAY_DESCRIPTOR *pArrayDescriptor, CUarray hArray)

 Get a 1D or 2D CUDA array descriptor.
- CUresult cuMemAlloc (CUdeviceptr *dptr, unsigned int bytesize)

Allocates device memory.

• CUresult cuMemAllocHost (void **pp, unsigned int bytesize)

Allocates page-locked host memory.

• CUresult cuMemAllocPitch (CUdeviceptr *dptr, unsigned int *pPitch, unsigned int WidthInBytes, unsigned int Height, unsigned int ElementSizeBytes)

Allocates pitched device memory.

CUresult cuMemcpy2D (const CUDA_MEMCPY2D *pCopy)

Copies memory for 2D arrays.

CUresult cuMemcpy2DAsync (const CUDA_MEMCPY2D *pCopy, CUstream hStream)

Copies memory for 2D arrays.

• CUresult cuMemcpy2DUnaligned (const CUDA_MEMCPY2D *pCopy)

Copies memory for 2D arrays.

• CUresult cuMemcpy3D (const CUDA_MEMCPY3D *pCopy)

Copies memory for 3D arrays.

• CUresult cuMemcpy3DAsync (const CUDA_MEMCPY3D *pCopy, CUstream hStream)

Copies memory for 3D arrays.

CUresult cuMemcpyAtoA (CUarray dstArray, unsigned int dstIndex, CUarray srcArray, unsigned int srcIndex, unsigned int ByteCount)

Copies memory from Array to Array.

CUresult cuMemcpyAtoD (CUdeviceptr dstDevice, CUarray hSrc, unsigned int SrcIndex, unsigned int Byte-Count)

Copies memory from Array to Device.

- CUresult cuMemcpyAtoH (void *dstHost, CUarray srcArray, unsigned int srcIndex, unsigned int ByteCount)
 Copies memory from Array to Host.
- CUresult cuMemcpyAtoHAsync (void *dstHost, CUarray srcArray, unsigned int srcIndex, unsigned int Byte-Count, CUstream hStream)

Copies memory from Array to Host.

 CUresult cuMemcpyDtoA (CUarray dstArray, unsigned int dstIndex, CUdeviceptr srcDevice, unsigned int Byte-Count)

Copies memory from Device to Array.

- CUresult cuMemcpyDtoD (CUdeviceptr dstDevice, CUdeviceptr srcDevice, unsigned int ByteCount)
 Copies memory from Device to Device.
- CUresult cuMemcpyDtoDAsync (CUdeviceptr dstDevice, CUdeviceptr srcDevice, unsigned int ByteCount, CUstream hStream)

Copies memory from Device to Device.

- CUresult cuMemcpyDtoH (void *dstHost, CUdeviceptr srcDevice, unsigned int ByteCount)
 Copies memory from Device to Host.
- CUresult cuMemcpyDtoHAsync (void *dstHost, CUdeviceptr srcDevice, unsigned int ByteCount, CUstream hStream)

Copies memory from Device to Host.

 CUresult cuMemcpyHtoA (CUarray dstArray, unsigned int dstIndex, const void *pSrc, unsigned int Byte-Count)

Copies memory from Host to Array.

CUresult cuMemcpyHtoAAsync (CUarray dstArray, unsigned int dstIndex, const void *pSrc, unsigned int Byte-Count, CUstream hStream)

Copies memory from Host to Array.

- CUresult cuMemcpyHtoD (CUdeviceptr dstDevice, const void *srcHost, unsigned int ByteCount)
 Copies memory from Host to Device.
- CUresult cuMemcpyHtoDAsync (CUdeviceptr dstDevice, const void *srcHost, unsigned int ByteCount, CUstream hStream)

Copies memory from Host to Device.

• CUresult cuMemFree (CUdeviceptr dptr)

Frees device memory.

CUresult cuMemFreeHost (void *p)

Frees page-locked host memory.

CUresult cuMemGetAddressRange (CUdeviceptr *pbase, unsigned int *psize, CUdeviceptr dptr)

Get information on memory allocations.

• CUresult cuMemGetInfo (unsigned int *free, unsigned int *total)

Gets free and total memory.

• CUresult cuMemHostAlloc (void **pp, size_t bytesize, unsigned int Flags)

Allocates page-locked host memory.

• CUresult cuMemHostGetDevicePointer (CUdeviceptr *pdptr, void *p, unsigned int Flags)

Passes back device pointer of mapped pinned memory.

• CUresult cuMemHostGetFlags (unsigned int *pFlags, void *p)

Passes back flags that were used for a pinned allocation.

• CUresult cuMemsetD16 (CUdeviceptr dstDevice, unsigned short us, unsigned int N)

Initializes device memory.

• CUresult cuMemsetD2D16 (CUdeviceptr dstDevice, unsigned int dstPitch, unsigned short us, unsigned int Width, unsigned int Height)

Initializes device memory.

CUresult cuMemsetD2D32 (CUdeviceptr dstDevice, unsigned int dstPitch, unsigned int ui, unsigned int Width, unsigned int Height)

Initializes device memory.

CUresult cuMemsetD2D8 (CUdeviceptr dstDevice, unsigned int dstPitch, unsigned char uc, unsigned int Width, unsigned int Height)

Initializes device memory.

• CUresult cuMemsetD32 (CUdeviceptr dstDevice, unsigned int ui, unsigned int N)

Initializes device memory.

• CUresult cuMemsetD8 (CUdeviceptr dstDevice, unsigned char uc, unsigned int N)

Initializes device memory.

4.30.1 Detailed Description

This section describes the memory management functions of the low-level CUDA driver application programming interface.

4.30.2 Function Documentation

4.30.2.1 cuArray3DCreate (CUarray * pHandle, const CUDA_ARRAY3D_DESCRIPTOR * pAllocateArray)

Creates a CUDA array according to the CUDA_ARRAY3D_DESCRIPTOR structure pallocateArray and returns a handle to the new CUDA array in *pHandle. The CUDA_ARRAY3D_DESCRIPTOR is defined as:

```
typedef struct {
   unsigned int Width;
   unsigned int Height;
   unsigned int Depth;
   CUarray_format Format;
```

```
unsigned int NumChannels;
unsigned int Flags;
} CUDA_ARRAY3D_DESCRIPTOR;
```

where:

- Width, Height, and Depth are the width, height, and depth of the CUDA array (in elements); the CUDA array is one-dimensional if height and depth are 0, two-dimensional if depth is 0, and three-dimensional otherwise;
- Format specifies the format of the elements; CUarray format is defined as:

```
typedef enum CUarray_format_enum {
   CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
   CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
   CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
   CU_AD_FORMAT_SIGNED_INT8 = 0x08,
   CU_AD_FORMAT_SIGNED_INT16 = 0x09,
   CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
   CU_AD_FORMAT_HALF = 0x10,
   CU_AD_FORMAT_FLOAT = 0x20
} CUarray_format;
```

- NumChannels specifies the number of packed components per CUDA array element; it may be 1, 2, or 4;
- Flags provides for future features. For now, it must be set to 0.

Here are examples of CUDA array descriptions:

Description for a CUDA array of 2048 floats:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 2048;
desc.Height = 0;
desc.Depth = 0;
```

Description for a 64 x 64 CUDA array of floats:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 64;
desc.Height = 64;
desc.Depth = 0;
```

Description for a width x height x depth CUDA array of 64-bit, 4x16-bit float16's:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.FormatFlags = CU_AD_FORMAT_HALF;
desc.NumChannels = 4;
desc.Width = width;
desc.Height = height;
desc.Depth = depth;
```

Parameters:

```
pHandle - Returned arraypAllocateArray - 3D array descriptor
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemCpyHtoAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.2 cuArray3DGetDescriptor (CUDA_ARRAY3D_DESCRIPTOR * pArrayDescriptor, CUarray hArray)

Returns in *pArrayDescriptor a descriptor containing information on the format and dimensions of the CUDA array hArray. It is useful for subroutines that have been passed a CUDA array, but need to know the CUDA array parameters for validation or other purposes.

This function may be called on 1D and 2D arrays, in which case the Height and/or Depth members of the descriptor struct will be set to 0.

Parameters:

```
pArrayDescriptor - Returned 3D array descriptorhArray - 3D array to get descriptor of
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.3 cuArrayCreate (CUarray * pHandle, const CUDA_ARRAY_DESCRIPTOR * pAllocateArray)

Creates a CUDA array according to the CUDA_ARRAY_DESCRIPTOR structure pAllocateArray and returns a handle to the new CUDA array in *pHandle. The CUDA_ARRAY_DESCRIPTOR is defined as:

```
typedef struct {
   unsigned int Width;
   unsigned int Height;
   CUarray_format Format;
   unsigned int NumChannels;
} CUDA_ARRAY_DESCRIPTOR;
```

where:

- Width, and Height are the width, and height of the CUDA array (in elements); the CUDA array is onedimensional if height is 0, two-dimensional otherwise;
- Format specifies the format of the elements; CUarray_format is defined as:

```
typedef enum CUarray_format_enum {
   CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
   CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
   CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
   CU_AD_FORMAT_SIGNED_INT8 = 0x08,
   CU_AD_FORMAT_SIGNED_INT16 = 0x09,
   CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
   CU_AD_FORMAT_HALF = 0x10,
   CU_AD_FORMAT_HALF = 0x20
} CUarray_format;
```

• NumChannels specifies the number of packed components per CUDA array element; it may be 1, 2, or 4;

Here are examples of CUDA array descriptions:

Description for a CUDA array of 2048 floats:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 2048;
desc.Height = 1;
```

Description for a 64 x 64 CUDA array of floats:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 64;
desc.Height = 64;
```

Description for a width x height CUDA array of 64-bit, 4x16-bit float16's:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.FormatFlags = CU_AD_FORMAT_HALF;
desc.NumChannels = 4;
desc.Width = width;
desc.Height = height;
```

Description for a width x height CUDA array of 16-bit elements, each of which is two 8-bit unsigned chars:

```
CUDA_ARRAY_DESCRIPTOR arrayDesc;
desc.FormatFlags = CU_AD_FORMAT_UNSIGNED_INT8;
desc.NumChannels = 2;
desc.Width = width;
desc.Height = height;
```

Parameters:

```
pHandle - Returned array
pAllocateArray - Array descriptor
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.4 cuArrayDestroy (CUarray hArray)

Destroys the CUDA array hArray.

Parameters:

hArray - Array to destroy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ARRAY_IS_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAAsync, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.5 cuArrayGetDescriptor (CUDA_ARRAY_DESCRIPTOR * pArrayDescriptor, CUarray hArray)

Returns in *pArrayDescriptor a descriptor containing information on the format and dimensions of the CUDA array hArray. It is useful for subroutines that have been passed a CUDA array, but need to know the CUDA array parameters for validation or other purposes.

Parameters:

```
pArrayDescriptor - Returned array descriptorhArray - Array to get descriptor of
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.6 cuMemAlloc (CUdeviceptr * dptr, unsigned int bytesize)

Allocates bytesize bytes of linear memory on the device and returns in *dptr a pointer to the allocated memory. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. If bytesize is 0, cuMemAlloc() returns CUDA_ERROR_INVALID_VALUE.

Parameters:

```
dptr - Returned device pointerbytesize - Requested allocation size in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA,

cuMemcpyHtoAAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.7 cuMemAllocHost (void ** pp, unsigned int bytesize)

Allocates bytesize bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as cuMemcpy(). Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as malloc(). Allocating excessive amounts of memory with cuMemAllocHost() may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

Parameters:

pp - Returned host pointer to page-locked memorybytesize - Requested allocation size in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemCpyHtoAsync, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.8 cuMemAllocPitch (CUdeviceptr * dptr, unsigned int * pPitch, unsigned int WidthInBytes, unsigned int Height, unsigned int ElementSizeBytes)

Allocates at least WidthInBytes * Height bytes of linear memory on the device and returns in *dptr a pointer to the allocated memory. The function may pad the allocation to ensure that corresponding pointers in any given row will continue to meet the alignment requirements for coalescing as the address is updated from row to row. ElementSizeBytes specifies the size of the largest reads and writes that will be performed on the memory range. ElementSizeBytes may be 4, 8 or 16 (since coalesced memory transactions are not possible on other data sizes). If ElementSizeBytes is smaller than the actual read/write size of a kernel, the kernel will run correctly, but possibly at reduced speed. The pitch returned in *pPitch by cuMemAllocPitch() is the width in bytes of the allocation. The intended usage of pitch is as a separate parameter of the allocation, used to compute addresses within the 2D array. Given the row and column of an array element of type T, the address is computed as:

```
T* pElement = (T*)((char*)BaseAddress + Row * Pitch) + Column;
```

The pitch returned by cuMemAllocPitch() is guaranteed to work with cuMemcpy2D() under all circumstances. For allocations of 2D arrays, it is recommended that programmers consider performing pitch allocations using cuMemAllocPitch(). Due to alignment restrictions in the hardware, this is especially true if the application will be performing 2D memory copies between different regions of device memory (whether linear memory or CUDA arrays).

The byte alignment of the pitch returned by cuMemAllocPitch() is guaranteed to match or exceed the alignment requirement for texture binding with cuTexRefSetAddress2D().

Parameters:

```
    dptr - Returned device pointer
    pPitch - Returned pitch of allocation in bytes
    WidthInBytes - Requested allocation width in bytes
    Height - Requested allocation height in rows
    ElementSizeBytes - Size of largest reads/writes for range
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.9 cuMemcpy2D (const CUDA MEMCPY2D * pCopy)

Perform a 2D memory copy according to the parameters specified in pCopy. The CUDA_MEMCPY2D structure is defined as:

```
typedef struct CUDA_MEMCPY2D_st {
   unsigned int srcXInBytes, srcY;
   CUmemorytype srcMemoryType;
       const void *srcHost;
       CUdeviceptr srcDevice;
       CUarray srcArray;
       unsigned int srcPitch;
   unsigned int dstXInBytes, dstY;
   CUmemorytype dstMemoryType;
       void *dstHost;
       CUdeviceptr dstDevice;
       CUarray dstArray;
       unsigned int dstPitch;
   unsigned int WidthInBytes;
  unsigned int Height;
} CUDA_MEMCPY2D;
```

where:

• srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
   CU_MEMORYTYPE_HOST = 0x01,
   CU_MEMORYTYPE_DEVICE = 0x02,
   CU_MEMORYTYPE_ARRAY = 0x03
} CUmemorytype;
```

If srcMemoryType is CU_MEMORYTYPE_HOST, srcHost and srcPitch specify the (host) base address of the source data and the bytes per row to apply. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_DEVICE, srcDevice and srcPitch specify the (device) base address of the source data and the bytes per row to apply. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_ARRAY, srcArray specifies the handle of the source data. srcHost, srcDevice and srcPitch are ignored.

If dstMemoryType is CU_MEMORYTYPE_HOST, dstHost and dstPitch specify the (host) base address of the destination data and the bytes per row to apply. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_DEVICE, dstDevice and dstPitch specify the (device) base address of the destination data and the bytes per row to apply. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_ARRAY, dstArray specifies the handle of the destination data. dstHost, dstDevice and dstPitch are ignored.

• srcXInBytes and srcY specify the base address of the source data for the copy.

For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
```

For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

• dstXInBytes and dstY specify the base address of the destination data for the copy.

For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;
```

For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

• WidthInBytes and Height specify the width (in bytes) and height of the 2D copy being performed. Any pitches must be greater than or equal to WidthInBytes.

cuMemcpy2D() returns an error if any pitch is greater than the maximum allowed (CU_DEVICE_ATTRIBUTE_-MAX_PITCH). cuMemAllocPitch() passes back pitches that always work with cuMemcpy2D(). On intra-device memory copies (device? device, CUDA array? device, CUDA array? CUDA array), cuMemcpy2D() may fail for pitches not computed by cuMemAllocPitch(). cuMemcpy2DUnaligned() does not have this restriction, but may run significantly slower in the cases where cuMemcpy2D() would have returned an error code.

Parameters:

```
pCopy - Parameters for the memory copy
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoB, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAAsync, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.10 cuMemcpy2DAsync (const CUDA_MEMCPY2D * pCopy, CUstream hStream)

Perform a 2D memory copy according to the parameters specified in pCopy. The CUDA_MEMCPY2D structure is defined as:

```
typedef struct CUDA_MEMCPY2D_st {
   unsigned int srcXInBytes, srcY;
   CUmemorytype srcMemoryType;
   const void *srcHost;
   CUdeviceptr srcDevice;
   CUarray srcArray;
   unsigned int srcPitch;
   unsigned int dstXInBytes, dstY;
   CUmemorytype dstMemoryType;
   void *dstHost;
   CUdeviceptr dstDevice;
```

```
CUarray dstArray;
unsigned int dstPitch;
unsigned int WidthInBytes;
unsigned int Height;
} CUDA_MEMCPY2D;
```

where:

• srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
   CU_MEMORYTYPE_HOST = 0x01,
   CU_MEMORYTYPE_DEVICE = 0x02,
   CU_MEMORYTYPE_ARRAY = 0x03
} CUmemorytype;
```

If srcMemoryType is CU_MEMORYTYPE_HOST, srcHost and srcPitch specify the (host) base address of the source data and the bytes per row to apply. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_DEVICE, srcDevice and srcPitch specify the (device) base address of the source data and the bytes per row to apply. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_ARRAY, srcArray specifies the handle of the source data. srcHost, srcDevice and srcPitch are ignored.

If dstMemoryType is CU_MEMORYTYPE_HOST, dstHost and dstPitch specify the (host) base address of the destination data and the bytes per row to apply. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_DEVICE, dstDevice and dstPitch specify the (device) base address of the destination data and the bytes per row to apply. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_ARRAY, dstArray specifies the handle of the destination data. dstHost, dstDevice and dstPitch are ignored.

• srcXInBytes and srcY specify the base address of the source data for the copy.

For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
```

For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

dstXInBytes and dstY specify the base address of the destination data for the copy.

For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;
```

For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

• WidthInBytes and Height specify the width (in bytes) and height of the 2D copy being performed. Any pitches must be greater than or equal to WidthInBytes.

cuMemcpy2D() returns an error if any pitch is greater than the maximum allowed (CU_DEVICE_ATTRIBUTE_-MAX_PITCH). cuMemAllocPitch() passes back pitches that always work with cuMemcpy2D(). On intra-device memory copies (device? device, CUDA array? device, CUDA array? CUDA array), cuMemcpy2D() may fail for pitches not computed by cuMemAllocPitch(). cuMemcpy2DUnaligned() does not have this restriction, but may run significantly slower in the cases where cuMemcpy2D() would have returned an error code.

cuMemcpy2DAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero hStream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

Parameters:

```
pCopy - Parameters for the memory copyhStream - Stream identifier
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.11 cuMemcpy2DUnaligned (const CUDA_MEMCPY2D * pCopy)

Perform a 2D memory copy according to the parameters specified in pCopy. The CUDA_MEMCPY2D structure is defined as:

```
typedef struct CUDA_MEMCPY2D_st {
   unsigned int srcXInBytes, srcY;
   CUmemorytype srcMemoryType;
   const void *srcHost;
  CUdeviceptr srcDevice;
  CUarray srcArray;
  unsigned int srcPitch;
  unsigned int dstXInBytes, dstY;
  CUmemorytype dstMemoryType;
  void *dstHost;
  CUdeviceptr dstDevice;
  CUarray dstArray;
  unsigned int dstPitch;
  unsigned int WidthInBytes;
  unsigned int Height;
} CUDA_MEMCPY2D;
```

where:

• srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
   CU_MEMORYTYPE_HOST = 0x01,
   CU_MEMORYTYPE_DEVICE = 0x02,
   CU_MEMORYTYPE_ARRAY = 0x03
} CUmemorytype;
```

If srcMemoryType is CU_MEMORYTYPE_HOST, srcHost and srcPitch specify the (host) base address of the source data and the bytes per row to apply. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_DEVICE, srcDevice and srcPitch specify the (device) base address of the source data and the bytes per row to apply. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_ARRAY, srcArray specifies the handle of the source data. srcHost, srcDevice and srcPitch are ignored.

If dstMemoryType is CU_MEMORYTYPE_HOST, dstHost and dstPitch specify the (host) base address of the destination data and the bytes per row to apply. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_DEVICE, dstDevice and dstPitch specify the (device) base address of the destination data and the bytes per row to apply. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_ARRAY, dstArray specifies the handle of the destination data. dstHost, dstDevice and dstPitch are ignored.

• srcXInBytes and srcY specify the base address of the source data for the copy.

For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
```

For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

dstXInBytes and dstY specify the base address of the destination data for the copy.

For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;
```

For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

• WidthInBytes and Height specify the width (in bytes) and height of the 2D copy being performed. Any pitches must be greater than or equal to WidthInBytes.

cuMemcpy2D() returns an error if any pitch is greater than the maximum allowed (CU_DEVICE_ATTRIBUTE_-MAX_PITCH). cuMemAllocPitch() passes back pitches that always work with cuMemcpy2D(). On intra-device memory copies (device? device, CUDA array? device, CUDA array? CUDA array), cuMemcpy2D() may fail for pitches not computed by cuMemAllocPitch(). cuMemcpy2DUnaligned() does not have this restriction, but may run significantly slower in the cases where cuMemcpy2D() would have returned an error code.

Parameters:

pCopy - Parameters for the memory copy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoB, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFree, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.12 cuMemcpy3D (const CUDA_MEMCPY3D * pCopy)

Perform a 3D memory copy according to the parameters specified in pCopy. The CUDA_MEMCPY3D structure is defined as:

```
typedef struct CUDA_MEMCPY3D_st {
   unsigned int srcXInBytes, srcY, srcZ;
   unsigned int srcLOD;
   CUmemorytype srcMemoryType;
       const void *srcHost;
        CUdeviceptr srcDevice;
       CUarray srcArray;
       unsigned int srcPitch; // ignored when src is array
       unsigned int srcHeight; // ignored when src is array; may be 0 if Depth==1
   unsigned int dstXInBytes, dstY, dstZ;
   unsigned int dstLOD;
    CUmemorytype dstMemoryType;
       void *dstHost;
       CUdeviceptr dstDevice;
       CUarray dstArray;
       unsigned int dstPitch; // ignored when dst is array
       unsigned int dstHeight; // ignored when dst is array; may be 0 if Depth==1
   unsigned int WidthInBytes;
   unsigned int Height;
   unsigned int Depth;
} CUDA_MEMCPY3D;
```

where:

• srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
   CU_MEMORYTYPE_HOST = 0x01,
   CU_MEMORYTYPE_DEVICE = 0x02,
   CU_MEMORYTYPE_ARRAY = 0x03
} CUmemorytype;
```

If srcMemoryType is CU_MEMORYTYPE_HOST, srcHost, srcPitch and srcHeight specify the (host) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_DEVICE, srcDevice, srcPitch and srcHeight specify the (device) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_ARRAY, srcArray specifies the handle of the source data. srcHost, srcDevice, srcPitch and srcHeight are ignored.

If dstMemoryType is CU_MEMORYTYPE_HOST, dstHost and dstPitch specify the (host) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_DEVICE, dstDevice and dstPitch specify the (device) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_ARRAY, dstArray specifies the handle of the destination data. dstHost, dstDevice, dstPitch and dstHeight are ignored.

• srcXInBytes, srcY and srcZ specify the base address of the source data for the copy.

For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+(srcZ*srcHeight+srcY)*srcPitch + srcXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+(srcZ*srcHeight+srcY)*srcPitch+srcXInBytes;
```

For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

dstXInBytes, dstY and dstZ specify the base address of the destination data for the copy.

For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+(dstZ*dstHeight+dstY)*dstPitch + dstXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+(dstZ*dstHeight+dstY)*dstPitch+dstXInBytes;
```

For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

• WidthInBytes, Height and Depth specify the width (in bytes), height and depth of the 3D copy being performed. Any pitches must be greater than or equal to WidthInBytes.

cuMemcpy3D() returns an error if any pitch is greater than the maximum allowed (CU_DEVICE_ATTRIBUTE_-MAX_PITCH).

The srcLOD and dstLOD members of the CUDA_MEMCPY3D structure must be set to 0.

Parameters:

```
pCopy - Parameters for the memory copy
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.13 cuMemcpy3DAsync (const CUDA_MEMCPY3D * pCopy, CUstream hStream)

Perform a 3D memory copy according to the parameters specified in pCopy. The CUDA_MEMCPY3D structure is defined as:

```
typedef struct CUDA_MEMCPY3D_st {
   unsigned int srcXInBytes, srcY, srcZ;
   unsigned int srcLOD;
   CUmemorytype srcMemoryType;
       const void *srcHost;
       CUdeviceptr srcDevice;
       CUarray srcArray;
       unsigned int srcPitch; // ignored when src is array
       unsigned int srcHeight; // ignored when src is array; may be 0 if Depth==1
   unsigned int dstXInBytes, dstY, dstZ;
   unsigned int dstLOD;
   CUmemorytype dstMemoryType;
       void *dstHost;
       CUdeviceptr dstDevice;
       CUarray dstArray;
       unsigned int dstPitch; // ignored when dst is array
       unsigned int dstHeight; // ignored when dst is array; may be 0 if Depth==1
   unsigned int WidthInBytes;
   unsigned int Height;
   unsigned int Depth;
} CUDA_MEMCPY3D;
```

where:

• srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
   CU_MEMORYTYPE_HOST = 0x01,
   CU_MEMORYTYPE_DEVICE = 0x02,
   CU_MEMORYTYPE_ARRAY = 0x03
} CUmemorytype;
```

If srcMemoryType is CU_MEMORYTYPE_HOST, srcHost, srcPitch and srcHeight specify the (host) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_DEVICE, srcDevice, srcPitch and srcHeight specify the (device) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

If srcMemoryType is CU_MEMORYTYPE_ARRAY, srcArray specifies the handle of the source data. srcHost, srcDevice, srcPitch and srcHeight are ignored.

If dstMemoryType is CU_MEMORYTYPE_HOST, dstHost and dstPitch specify the (host) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_DEVICE, dstDevice and dstPitch specify the (device) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

If dstMemoryType is CU_MEMORYTYPE_ARRAY, dstArray specifies the handle of the destination data. dstHost, dstDevice, dstPitch and dstHeight are ignored.

• srcXInBytes, srcY and srcZ specify the base address of the source data for the copy.

For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+(srcZ*srcHeight+srcY)*srcPitch + srcXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+(srcZ*srcHeight+srcY)*srcPitch+srcXInBytes;
```

For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

• dstXInBytes, dstY and dstZ specify the base address of the destination data for the copy.

For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+(dstZ*dstHeight+dstY)*dstPitch + dstXInBytes);
```

For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+(dstZ*dstHeight+dstY)*dstPitch+dstXInBytes;
```

For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

• WidthInBytes, Height and Depth specify the width (in bytes), height and depth of the 3D copy being performed. Any pitches must be greater than or equal to WidthInBytes.

cuMemcpy3D() returns an error if any pitch is greater than the maximum allowed (CU_DEVICE_ATTRIBUTE_-MAX_PITCH).

cuMemcpy3DAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero hStream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

The srcLOD and dstLOD members of the CUDA_MEMCPY3D structure must be set to 0.

Parameters:

pCopy - Parameters for the memory copy*hStream* - Stream identifier

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.14 cuMemcpyAtoA (CUarray dstArray, unsigned int dstIndex, CUarray srcArray, unsigned int srcIndex, unsigned int ByteCount)

Copies from one 1D CUDA array to another. dstArray and srcArray specify the handles of the destination and source CUDA arrays for the copy, respectively. dstIndex and srcIndex specify the destination and source indices into the CUDA array. These values are in the range [0, Width-1] for the CUDA array; they are not byte offsets. ByteCount is the number of bytes to be copied. The size of the elements in the CUDA arrays need not be the same format, but the elements must be the same size; and count must be evenly divisible by that size.

Parameters:

dstArray - Destination array

dstIndex - Offset of destination array

srcArray - Source array

srcIndex - Offset of source array

ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned,

cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.15 cuMemcpyAtoD (CUdeviceptr dstDevice, CUarray hSrc, unsigned int SrcIndex, unsigned int ByteCount)

Copies from one 1D CUDA array to device memory. dstDevice specifies the base pointer of the destination and must be naturally aligned with the CUDA array elements. hSrc and SrcIndex specify the CUDA array handle and the index (in array elements) of the array element where the copy is to begin. ByteCount specifies the number of bytes to copy and must be evenly divisible by the array element size.

Parameters:

```
    dstDevice - Destination device pointer
    hSrc - Source array
    SrcIndex - Offset of source array
    ByteCount - Size of memory copy in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAAsync, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.16 cuMemcpyAtoH (void * dstHost, CUarray srcArray, unsigned int srcIndex, unsigned int ByteCount)

Copies from one 1D CUDA array to host memory. dstHost specifies the base pointer of the destination. srcArray and srcIndex specify the CUDA array handle and starting index of the source data. ByteCount specifies the number of bytes to copy.

Parameters:

```
dstHost - Destination device pointersrcArray - Source arraysrcIndex - Offset of source array
```

ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAAsync, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.17 cuMemcpyAtoHAsync (void * dstHost, CUarray srcArray, unsigned int srcIndex, unsigned int ByteCount, CUstream hStream)

Copies from one 1D CUDA array to host memory. dstHost specifies the base pointer of the destination. srcArray and srcIndex specify the CUDA array handle and starting index of the source data. ByteCount specifies the number of bytes to copy.

cuMemcpyAtoHAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero stream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

Parameters:

dstHost - Destination pointer

srcArray - Source array

srcIndex - Offset of source array

ByteCount - Size of memory copy in bytes

hStream - Stream identifier

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemCpyHtoAsync, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.18 cuMemcpyDtoA (CUarray dstArray, unsigned int dstIndex, CUdeviceptr srcDevice, unsigned int ByteCount)

Copies from device memory to a 1D CUDA array. dstArray and dstIndex specify the CUDA array handle and starting index of the destination data. srcDevice specifies the base pointer of the source. ByteCount specifies the number of bytes to copy.

Parameters:

dstArray - Destination array
 dstIndex - Offset of destination array
 srcDevice - Source device pointer
 ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.19 cuMemcpyDtoD (CUdeviceptr dstDevice, CUdeviceptr srcDevice, unsigned int ByteCount)

Copies from device memory to device memory. dstDevice and srcDevice are the base pointers of the destination and source, respectively. ByteCount specifies the number of bytes to copy. Note that this function is asynchronous.

Parameters:

dstDevice - Destination device pointersrcDevice - Source device pointerByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAasync, cuMemcpyHtoD, cuMemcpyHtoDasync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD32

4.30.2.20 cuMemcpyDtoDAsync (CUdeviceptr dstDevice, CUdeviceptr srcDevice, unsigned int ByteCount, CUstream hStream)

Copies from device memory to device memory. dstDevice and srcDevice are the base pointers of the destination and source, respectively. ByteCount specifies the number of bytes to copy. Note that this function is asynchronous and can optionally be associated to a stream by passing a non-zero hStream argument

Parameters:

dstDevice - Destination device pointer
 srcDevice - Source device pointer
 ByteCount - Size of memory copy in bytes
 hStream - Stream identifier

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoD, cuMemcpyDtoH, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.21 cuMemcpyDtoH (void * dstHost, CUdeviceptr srcDevice, unsigned int ByteCount)

Copies from device to host memory. dstHost and srcDevice specify the base pointers of the destination and source, respectively. ByteCount specifies the number of bytes to copy. Note that this function is synchronous.

Parameters:

dstHost - Destination host pointersrcDevice - Source device pointerByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.22 cuMemcpyDtoHAsync (void * dstHost, CUdeviceptr srcDevice, unsigned int ByteCount, CUstream hStream)

Copies from device to host memory. dstHost and srcDevice specify the base pointers of the destination and source, respectively. ByteCount specifies the number of bytes to copy.

cuMemcpyDtoHAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero hStream argument. It only works on page-locked memory and returns an error if a pointer to pageable memory is passed as input.

Parameters:

dstHost - Destination host pointer
 srcDevice - Source device pointer
 ByteCount - Size of memory copy in bytes
 hStream - Stream identifier

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoA, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.23 cuMemcpyHtoA (CUarray dstArray, unsigned int dstIndex, const void * pSrc, unsigned int ByteCount)

Copies from host memory to a 1D CUDA array. dstArray and dstIndex specify the CUDA array handle and starting index of the destination data. pSrc specifies the base address of the source. ByteCount specifies the number of bytes to copy.

Parameters:

```
    dstArray - Destination array
    dstIndex - Offset of destination array
    pSrc - Source host pointer
    ByteCount - Size of memory copy in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoAasync, cuMemcpyHtoAasync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.24 cuMemcpyHtoAAsync (CUarray dstArray, unsigned int dstIndex, const void * pSrc, unsigned int ByteCount, CUstream hStream)

Copies from host memory to a 1D CUDA array. dstArray and dstIndex specify the CUDA array handle and starting index of the destination data. pSrc specifies the base address of the source. ByteCount specifies the number of bytes to copy.

cuMemcpyHtoAAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero hStream argument. It only works on page-locked memory and returns an error if a pointer to pageable memory is passed as input.

Parameters:

dstArray - Destination array
 dstIndex - Offset of destination array
 pSrc - Source host pointer
 ByteCount - Size of memory copy in bytes
 hStream - Stream identifier

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoD, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.25 cuMemcpyHtoD (CUdeviceptr dstDevice, const void * srcHost, unsigned int ByteCount)

Copies from host memory to device memory. dstDevice and srcHost are the base addresses of the destination and source, respectively. ByteCount specifies the number of bytes to copy. Note that this function is synchronous.

Parameters:

```
dstDevice - Destination device pointersrcHost - Source host pointerByteCount - Size of memory copy in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAasync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.26 cuMemcpyHtoDAsync (CUdeviceptr dstDevice, const void * srcHost, unsigned int ByteCount, CUstream hStream)

Copies from host memory to device memory. dstDevice and srcHost are the base addresses of the destination and source, respectively. ByteCount specifies the number of bytes to copy.

cuMemcpyHtoDAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero hStream argument. It only works on page-locked memory and returns an error if a pointer to pageable memory is passed as input.

Parameters:

dstDevice - Destination device pointer
 srcHost - Source host pointer
 ByteCount - Size of memory copy in bytes
 hStream - Stream identifier

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoA, cuMemcpyHtoA, cuMemcpyHtoD, cuMemFree, cuMemFreeHost, cuMemGetAddress-Range, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.27 cuMemFree (CUdeviceptr dptr)

Frees the memory space pointed to by dptr, which must have been returned by a previous call to cuMemAlloc() or cuMemAllocPitch().

Parameters:

dptr - Pointer to memory to free

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoD, cuMemcpyHtoDAsync, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.28 cuMemFreeHost (void *p)

Frees the memory space pointed to by p, which must have been returned by a previous call to cuMemAllocHost().

Parameters:

p - Pointer to memory to free

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoD, cuMemcpyHtoDAsync, cuMemFree, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.29 cuMemGetAddressRange (CUdeviceptr * pbase, unsigned int * psize, CUdeviceptr dptr)

Returns the base address in *pbase and size in *psize of the allocation by cuMemAlloc() or cuMemAllocPitch() that contains the input pointer dptr. Both parameters pbase and psize are optional. If one of them is NULL, it is ignored.

Parameters:

```
pbase - Returned base addresspsize - Returned size of device memory allocationdptr - Device pointer to query
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFree-Host, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.30 cuMemGetInfo (unsigned int * free, unsigned int * total)

Returns in *free and *total respectively, the free and total amount of memory available for allocation by the CUDA context, in bytes.

Parameters:

free - Returned free memory in bytestotal - Returned total memory in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAasync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.31 cuMemHostAlloc (void ** pp, size t bytesize, unsigned int Flags)

Allocates bytesize bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as cuMemcpyHtoD(). Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as malloc(). Allocating excessive amounts of pinned memory may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

The Flags parameter enables different options to be specified that affect the allocation, as follows.

- CU_MEMHOSTALLOC_PORTABLE: The memory returned by this call will be considered as pinned memory by all CUDA contexts, not just the one that performed the allocation.
- CU_MEMHOSTALLOC_DEVICEMAP: Maps the allocation into the CUDA address space. The device pointer to the memory may be obtained by calling cuMemHostGetDevicePointer(). This feature is available only on GPUs with compute capability greater than or equal to 1.1.
- CU_MEMHOSTALLOC_WRITECOMBINED: Allocates the memory as write-combined (WC). WC memory can be transferred across the PCI Express bus more quickly on some system configurations, but cannot be read efficiently by most CPUs. WC memory is a good option for buffers that will be written by the CPU and read by the GPU via mapped pinned memory or host->device transfers.

All of these flags are orthogonal to one another: a developer may allocate memory that is portable, mapped and/or write-combined with no restrictions.

The CUDA context must have been created with the CU_CTX_MAP_HOST flag in order for the CU_MEMHOSTALLOC_MAPPED flag to have any effect.

The CU_MEMHOSTALLOC_MAPPED flag may be specified on CUDA contexts for devices that do not support mapped pinned memory. The failure is deferred to cuMemHostGetDevicePointer() because the memory may be mapped into other CUDA contexts via the CU_MEMHOSTALLOC_PORTABLE flag.

The memory allocated by this function must be freed with cuMemFreeHost().

Parameters:

```
pp - Returned host pointer to page-locked memorybytesize - Requested allocation size in bytesFlags - Flags for allocation request
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.32 cuMemHostGetDevicePointer (CUdeviceptr * pdptr, void * p, unsigned int Flags)

Passes back the device pointer pdptr corresponding to the mapped, pinned host buffer p allocated by cuMemHostAlloc.

cuMemHostGetDevicePointer() will fail if the CU_MEMALLOCHOST_DEVICEMAP flag was not specified at the time the memory was allocated, or if the function is called on a GPU that does not support mapped pinned memory.

Flags provides for future releases. For now, it must be set to 0.

Parameters:

```
pdptr - Returned device pointerp - Host pointerFlags - Options (must be 0)
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAasync, cuMemcpyHtoD, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.33 cuMemHostGetFlags (unsigned int * *pFlags*, void * *p*)

Passes back the flags pFlags that were specified when allocating the pinned host buffer p allocated by cuMemHostAlloc.

cuMemHostGetFlags() will fail if the pointer does not reside in an allocation performed by cuMemAllocHost() or cuMemHostAlloc().

Parameters:

```
pFlags - Returned flags wordp - Host pointer
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMemAllocHost, cuMemHostAlloc

4.30.2.34 cuMemsetD16 (CUdeviceptr dstDevice, unsigned short us, unsigned int N)

Sets the memory range of N 16-bit values to the specified value us.

Parameters:

```
dstDevice - Destination device pointerus - Value to setN - Number of elements
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD32

4.30.2.35 cuMemsetD2D16 (CUdeviceptr *dstDevice*, unsigned int *dstPitch*, unsigned short *us*, unsigned int *Width*, unsigned int *Height*)

Sets the 2D memory range of Width 16-bit values to the specified value us. Height specifies the number of rows to set, and dstPitch specifies the number of bytes between each row. This function performs fastest when the pitch is one that has been passed back by cuMemAllocPitch().

Parameters:

```
dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
us - Value to set
Width - Width of row
Height - Number of rows
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoA, cuMemcpyHtoA, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.36 cuMemsetD2D32 (CUdeviceptr dstDevice, unsigned int dstPitch, unsigned int ui, unsigned int Width, unsigned int Height)

Sets the 2D memory range of Width 32-bit values to the specified value ui. Height specifies the number of rows to set, and dstPitch specifies the number of bytes between each row. This function performs fastest when the pitch is one that has been passed back by cuMemAllocPitch().

Parameters:

dstDevice - Destination device pointer

```
    dstPitch - Pitch of destination device pointer
    ui - Value to set
    Width - Width of row
    Height - Number of rows
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.37 cuMemsetD2D8 (CUdeviceptr *dstDevice*, unsigned int *dstPitch*, unsigned char *uc*, unsigned int *Width*, unsigned int *Height*)

Sets the 2D memory range of Width 8-bit values to the specified value uc. Height specifies the number of rows to set, and dstPitch specifies the number of bytes between each row. This function performs fastest when the pitch is one that has been passed back by cuMemAllocPitch().

Parameters:

```
dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
uc - Value to set
Width - Width of row
Height - Number of rows
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16, cuMemsetD32

4.30.2.38 cuMemsetD32 (CUdeviceptr dstDevice, unsigned int ui, unsigned int N)

Sets the memory range of N 32-bit values to the specified value ui.

Parameters:

dstDevice - Destination device pointerui - Value to setN - Number of elements

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD8, cuMemsetD16

4.30.2.39 cuMemsetD8 (CUdeviceptr dstDevice, unsigned char uc, unsigned int N)

Sets the memory range of N 8-bit values to the specified value uc.

Parameters:

dstDevice - Destination device pointeruc - Value to setN - Number of elements

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID CONTEXT, CUDA_ERROR_INVALID VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, cuArray3DGetDescriptor, cuArrayCreate, cuArrayDestroy, cuArrayGetDescriptor, cuMemAlloc, cuMemAllocHost, cuMemAllocPitch, cuMemcpy2D, cuMemcpy2DAsync, cuMemcpy2DUnaligned, cuMemcpy3D, cuMemcpy3DAsync, cuMemcpyAtoA, cuMemcpyAtoD, cuMemcpyAtoH, cuMemcpyAtoHAsync, cuMemcpyDtoA, cuMemcpyDtoDAsync, cuMemcpyDtoDAsync, cuMemcpyDtoHAsync, cuMemcpyHtoA, cuMemcpyHtoAsync, cuMemcpyHtoDAsync, cuMemFree, cuMemFreeHost, cuMemGetAddressRange, cuMemGetInfo, cuMemHostAlloc, cuMemHostGetDevicePointer, cuMemsetD2D8, cuMemsetD2D16, cuMemsetD2D32, cuMemsetD16, cuMemsetD32

4.31 Texture Reference Management

Functions

• CUresult cuTexRefCreate (CUtexref *pTexRef)

Creates a texture reference.

• CUresult cuTexRefDestroy (CUtexref hTexRef)

Destroys a texture reference.

CUresult cuTexRefGetAddress (CUdeviceptr *pdptr, CUtexref hTexRef)

Gets the address associated with a texture reference.

CUresult cuTexRefGetAddressMode (CUaddress_mode *pam, CUtexref hTexRef, int dim)

Gets the addressing mode used by a texture reference.

CUresult cuTexRefGetArray (CUarray *phArray, CUtexref hTexRef)

Gets the array bound to a texture reference.

CUresult cuTexRefGetFilterMode (CUfilter_mode *pfm, CUtexref hTexRef)

Gets the filter-mode used by a texture reference.

CUresult cuTexRefGetFlags (unsigned int *pFlags, CUtexref hTexRef)

Gets the flags used by a texture reference.

CUresult cuTexRefGetFormat (CUarray_format *pFormat, int *pNumChannels, CUtexref hTexRef)

Gets the format used by a texture reference.

CUresult cuTexRefSetAddress (unsigned int *ByteOffset, CUtexref hTexRef, CUdeviceptr dptr, unsigned int bytes)

Binds an address as a texture reference.

CUresult cuTexRefSetAddress2D (CUtexref hTexRef, const CUDA_ARRAY_DESCRIPTOR *desc, CUdeviceptr dptr, unsigned int Pitch)

Binds an address as a 2D texture reference.

CUresult cuTexRefSetAddressMode (CUtexref hTexRef, int dim, CUaddress_mode am)

Sets the addressing mode for a texture reference.

• CUresult cuTexRefSetArray (CUtexref hTexRef, CUarray hArray, unsigned int Flags)

Binds an address as a texture reference.

CUresult cuTexRefSetFilterMode (CUtexref hTexRef, CUfilter_mode fm)

Sets the filtering mode for a texture reference.

• CUresult cuTexRefSetFlags (CUtexref hTexRef, unsigned int Flags)

Sets the flags for a texture reference.

• CUresult cuTexRefSetFormat (CUtexref hTexRef, CUarray_format fmt, int NumPackedComponents)

Sets the format for a texture reference.

4.31.1 Detailed Description

This section describes the texture reference management functions of the low-level CUDA driver application programming interface.

4.31.2 Function Documentation

4.31.2.1 cuTexRefCreate (CUtexref * pTexRef)

Creates a texture reference and returns its handle in *pTexRef. Once created, the application must call cuTexRefSetArray() or cuTexRefSetAddress() to associate the reference with allocated memory. Other texture reference functions are used to specify the format and interpretation (addressing, filtering, etc.) to be used when the memory is read through this texture reference. To associate the texture reference with a texture ordinal for a given function, the application should call cuParamSetTexRef().

Parameters:

pTexRef - Returned texture reference

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddressMode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.2 cuTexRefDestroy (CUtexref *hTexRef*)

Destroys the texture reference specified by hTexRef.

Parameters:

hTexRef - Texture reference to destroy

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.3 cuTexRefGetAddress (CUdeviceptr * pdptr, CUtexref hTexRef)

Returns in *pdptr the base address bound to the texture reference hTexRef, or returns CUDA_ERROR_-INVALID_VALUE if the texture reference is not bound to any device memory range.

Parameters:

pdptr - Returned device address*hTexRef* - Texture reference

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.4 cuTexRefGetAddressMode (CUaddress_mode * pam, CUtexref hTexRef, int dim)

Returns in *pam the addressing mode corresponding to the dimension dim of the texture reference hTexRef. Currently, the only valid value for dim are 0 and 1.

Parameters:

```
pam - Returned addressing modehTexRef - Texture referencedim - Dimension
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.5 cuTexRefGetArray (CUarray * phArray, CUtexref hTexRef)

Returns in *phArray the CUDA array bound to the texture reference hTexRef, or returns CUDA_ERROR_-INVALID_VALUE if the texture reference is not bound to any CUDA array.

Parameters:

```
phArray - Returned arrayhTexRef - Texture reference
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddressMode, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.6 cuTexRefGetFilterMode (CUfilter_mode * pfm, CUtexref hTexRef)

Returns in *pfm the filtering mode of the texture reference hTexRef.

Parameters:

```
pfm - Returned filtering modehTexRef - Texture reference
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddressMode, cuTexRefGetArray, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.7 cuTexRefGetFlags (unsigned int * pFlags, CUtexref hTexRef)

Returns in *pFlags the flags of the texture reference hTexRef.

Parameters:

```
pFlags - Returned flags
hTexRef - Texture reference
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddressMode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFormat

4.31.2.8 cuTexRefGetFormat (CUarray_format * pFormat, int * pNumChannels, CUtexref hTexRef)

Returns in *pFormat and *pNumChannels the format and number of components of the CUDA array bound to the texture reference hTexRef. If pFormat or pNumChannels is NULL, it will be ignored.

Parameters:

```
pFormat - Returned formatpNumChannels - Returned number of componentshTexRef - Texture reference
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetFilterMode, cuTexRefSetFilterMode, cuTexRefGetAddress, cuTexRefGetAddressMode, cuTexRefGetAddressMode, cuTexRefGetFilterMode, cuTexRefGetFilte

4.31.2.9 cuTexRefSetAddress (unsigned int * ByteOffset, CUtexref hTexRef, CUdeviceptr dptr, unsigned int bytes)

Binds a linear address range to the texture reference hTexRef. Any previous address or CUDA array state associated with the texture reference is superseded by this function. Any memory previously bound to hTexRef is unbound.

Since the hardware enforces an alignment requirement on texture base addresses, cuTexRefSetAddress() passes back a byte offset in *ByteOffset that must be applied to texture fetches in order to read from the desired memory. This offset must be divided by the texel size and passed to kernels that read from the texture so they can be applied to the tex1Dfetch() function.

If the device memory pointer was returned from cuMemAlloc(), the offset is guaranteed to be 0 and NULL may be passed as the ByteOffset parameter.

Parameters:

ByteOffset - Returned byte offsethTexRef - Texture reference to binddptr - Device pointer to bind

bytes - Size of memory to bind in bytes

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.10 cuTexRefSetAddress2D (CUtexref hTexRef, const CUDA_ARRAY_DESCRIPTOR * desc, CUdeviceptr dptr, unsigned int Pitch)

Binds a linear address range to the texture reference hTexRef. Any previous address or CUDA array state associated with the texture reference is superseded by this function. Any memory previously bound to hTexRef is unbound.

Using a tex2D() function inside a kernel requires a call to either cuTexRefSetArray() to bind the corresponding texture reference to an array, or cuTexRefSetAddress2D() to bind the texture reference to linear memory.

Function calls to cuTexRefSetFormat() cannot follow calls to cuTexRefSetAddress2D() for the same texture reference.

It is required that dptr be aligned to the appropriate hardware-specific texture alignment. You can query this value using the device attribute $CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT$. If an unaligned dptr is supplied, $CUDA_ERROR_INVALID_VALUE$ is returned.

Parameters:

hTexRef - Texture reference to bind

```
desc - Descriptor of CUDA arraydptr - Device pointer to bindPitch - Line pitch in bytes
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.11 cuTexRefSetAddressMode (CUtexref hTexRef, int dim, CUaddress_mode am)

Specifies the addressing mode am for the given dimension dim of the texture reference hTexRef. If dim is zero, the addressing mode is applied to the first parameter of the functions used to fetch from the texture; if dim is 1, the second, and so on. CUaddress_mode is defined as:

```
typedef enum CUaddress_mode_enum {
   CU_TR_ADDRESS_MODE_WRAP = 0,
   CU_TR_ADDRESS_MODE_CLAMP = 1,
   CU_TR_ADDRESS_MODE_MIRROR = 2,
} CUaddress mode;
```

Note that this call has no effect if hTexRef is bound to linear memory.

Parameters:

```
hTexRef - Texture referencedim - Dimensionam - Addressing mode to set
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.12 cuTexRefSetArray (CUtexref hTexRef, CUarray hArray, unsigned int Flags)

Binds the CUDA array harray to the texture reference hTexRef. Any previous address or CUDA array state associated with the texture reference is superseded by this function. Flags must be set to CU_TRSA_OVERRIDE_FORMAT. Any CUDA array previously bound to hTexRef is unbound.

Parameters:

```
hTexRef - Texture reference to bindhArray - Array to bindFlags - Options (must be CU_TRSA_OVERRIDE_FORMAT)
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.13 cuTexRefSetFilterMode (CUtexref hTexRef, CUfilter_mode fm)

Specifies the filtering mode fm to be used when reading memory through the texture reference hTexRef. CUfilter_mode_enum is defined as:

```
typedef enum CUfilter_mode_enum {
   CU_TR_FILTER_MODE_POINT = 0,
   CU_TR_FILTER_MODE_LINEAR = 1
} CUfilter_mode;
```

Note that this call has no effect if hTexRef is bound to linear memory.

Parameters:

```
hTexRef - Texture referencefm - Filtering mode to set
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID CONTEXT, CUDA ERROR INVALID VALUE
```

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFlags, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddressMode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.14 cuTexRefSetFlags (CUtexref hTexRef, unsigned int Flags)

Specifies optional flags via Flags to specify the behavior of data returned through the texture reference hTexRef. The valid flags are:

• CU_TRSF_READ_AS_INTEGER, which suppresses the default behavior of having the texture promote integer data to floating point data in the range [0, 1];

• CU_TRSF_NORMALIZED_COORDINATES, which suppresses the default behavior of having the texture coordinates range from [0, Dim) where Dim is the width or height of the CUDA array. Instead, the texture coordinates [0, 1.0) reference the entire breadth of the array dimension;

Parameters:

```
hTexRef - Texture referenceFlags - Optional flags to set
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFormat, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.31.2.15 cuTexRefSetFormat (CUtexref hTexRef, CUarray_format fmt, int NumPackedComponents)

Specifies the format of the data to be read by the texture reference hTexRef. fmt and NumPackedComponents are exactly analogous to the Format and NumChannels members of the CUDA_ARRAY_DESCRIPTOR structure: They specify the format of each component and the number of components per array element.

Parameters:

```
hTexRef - Texture referencefmt - Format to setNumPackedComponents - Number of components per array element
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

See also:

cuTexRefCreate, cuTexRefDestroy, cuTexRefSetAddress, cuTexRefSetAddress2D, cuTexRefSetAddressMode, cuTexRefSetArray, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefGetAddress, cuTexRefGetAddress-Mode, cuTexRefGetArray, cuTexRefGetFilterMode, cuTexRefGetFlags, cuTexRefGetFormat

4.32 OpenGL Interoperability

Modules

• OpenGL Interoperability [DEPRECATED]

Functions

• CUresult cuGLCtxCreate (CUcontext *pCtx, unsigned int Flags, CUdevice device)

Create a CUDA context for interoperability with OpenGL.

• CUresult cuGraphicsGLRegisterBuffer (CUgraphicsResource *pCudaResource, GLuint buffer, unsigned int Flags)

Registers an OpenGL buffer object.

CUresult cuGraphicsGLRegisterImage (CUgraphicsResource *pCudaResource, GLuint image, GLenum target, unsigned int Flags)

Register an OpenGL texture or renderbuffer object.

• CUresult cuWGLGetDevice (CUdevice *pDevice, HGPUNV hGpu)

Gets the CUDA device associated with hGpu.

4.32.1 Detailed Description

This section describes the OpenGL interoperability functions of the low-level CUDA driver application programming interface.

4.32.2 Function Documentation

4.32.2.1 cuGLCtxCreate (CUcontext * pCtx, unsigned int Flags, CUdevice device)

Creates a new CUDA context, initializes OpenGL interoperability, and associates the CUDA context with the calling thread. It must be called before performing any other OpenGL interoperability operations. It may fail if the needed OpenGL driver facilities are not available. For usage of the Flags parameter, see cuCtxCreate().

Parameters:

pCtx - Returned CUDA context

Flags - Options for CUDA context creation

device - Device on which to create the context

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxCreate, cuGLInit, cuGLMapBufferObject, cuGLRegisterBufferObject, cuGLUnmapBufferObject, cuGLUnmapBufferObjectAsync, cuGLUnmapBufferObjectAsync, cuGLSetBufferObjectMapFlags, cuWGLGetDevice

4.32.2.2 cuGraphicsGLRegisterBuffer (CUgraphicsResource *pCudaResource, GLuint buffer, unsigned int Flags)

Registers the buffer object specified by buffer for access by CUDA. A handle to the registered object is returned as pCudaResource. The map flags Flags specify the intended usage, as follows:

- CU_GRAPHICS_MAP_RESOURCE_FLAGS_NONE: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA. This is the default value.
- CU_GRAPHICS_MAP_RESOURCE_FLAGS_READ_ONLY: Specifies that CUDA will not write to this resource.
- CU_GRAPHICS_MAP_RESOURCE_FLAGS_WRITE_DISCARD: Specifies that CUDA will not read from
 this resource and will write over the entire contents of the resource, so none of the data previously stored in the
 resource will be preserved.

Parameters:

```
pCudaResource - Pointer to the returned object handlebuffer - name of buffer object to be registeredFlags - Map flags
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED, CUDA_ERROR INVALID CONTEXT,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGLCtxCreate, cuGraphicsUnregisterResource, cuGraphicsMapResources, cuGraphicsResourceGetMapped-Pointer

4.32.2.3 cuGraphicsGLRegisterImage (CUgraphicsResource * pCudaResource, GLuint image, GLenum target, unsigned int Flags)

Registers the texture or renderbuffer object specified by image for access by CUDA. target must match the type of the object. A handle to the registered object is returned as pCudaResource. The map flags Flags specify the intended usage, as follows:

- CU_GRAPHICS_MAP_RESOURCE_FLAGS_NONE: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA. This is the default value.
- CU_GRAPHICS_MAP_RESOURCE_FLAGS_READ_ONLY: Specifies that CUDA will not write to this resource.

CU_GRAPHICS_MAP_RESOURCE_FLAGS_WRITE_DISCARD: Specifies that CUDA will not read from
this resource and will write over the entire contents of the resource, so none of the data previously stored in the
resource will be preserved.

The following image classes are currently disallowed:

- Textures with borders
- Multisampled renderbuffers

Parameters:

```
pCudaResource - Pointer to the returned object handle
```

image - name of texture or renderbuffer object to be registered

target - Identifies the type of object specified by image, and must be one of GL_TEXTURE_2D, GL_TEXTURE_RECTANGLE, GL_TEXTURE_CUBE_MAP, GL_TEXTURE_3D, GL_TEXTURE_-2D_ARRAY, or GL_RENDERBUFFER.

Flags - Map flags

Returns:

CUDA_SUCCESS, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED, CUDA_ERROR_INVALID_CONTEXT,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuGLCtxCreate, \qquad cuGraphicsUnregisterResource, \qquad cuGraphicsMapResources, \qquad cuGraphicsSubResourceGetMappedArray$

4.32.2.4 cuWGLGetDevice (CUdevice * pDevice, HGPUNV hGpu)

Returns in *pDevice the CUDA device associated with a hGpu, if applicable.

Parameters:

```
pDevice - Device associated with hGpuhGpu - Handle to a GPU, as queried via WGL_NV_gpu_affinity()
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGLCtxCreate, cuGLInit, cuGLMapBufferObject, cuGLRegisterBufferObject, cuGLUnmapBufferObject, cuGLUnmapBufferObjectAsync, cuGLSetBufferObjectMapFlags

4.33 OpenGL Interoperability [DEPRECATED]

Functions

• CUresult cuGLInit (void)

Initializes OpenGL interoperability.

• CUresult cuGLMapBufferObject (CUdeviceptr *dptr, unsigned int *size, GLuint buffer)

Maps an OpenGL buffer object.

CUresult cuGLMapBufferObjectAsync (CUdeviceptr *dptr, unsigned int *size, GLuint buffer, CUstream hStream)

Maps an OpenGL buffer object.

• CUresult cuGLRegisterBufferObject (GLuint buffer)

Registers an OpenGL buffer object.

• CUresult cuGLSetBufferObjectMapFlags (GLuint buffer, unsigned int Flags)

Set the map flags for an OpenGL buffer object.

• CUresult cuGLUnmapBufferObject (GLuint buffer)

Unmaps an OpenGL buffer object.

• CUresult cuGLUnmapBufferObjectAsync (GLuint buffer, CUstream hStream)

Unmaps an OpenGL buffer object.

• CUresult cuGLUnregisterBufferObject (GLuint buffer)

Unregister an OpenGL buffer object.

4.33.1 Detailed Description

This section describes deprecated OpenGL interoperability functionality.

4.33.2 Function Documentation

4.33.2.1 cuGLInit (**void**)

Deprecated

This function is deprecated as of Cuda 3.0.

Initializes OpenGL interoperability. This function is deprecated and calling it is no longer required. It may fail if the needed OpenGL driver facilities are not available.

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGLCtxCreate, cuGLMapBufferObject, cuGLRegisterBufferObject, cuGLUnmapBufferObject, cuGLUnmapBufferObject, cuGLUnmapBufferObjectAsync, cuGLSetBufferObjectMapFlags, cuWGLGetDevice

4.33.2.2 cuGLMapBufferObject (CUdeviceptr * dptr, unsigned int * size, GLuint buffer)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the buffer object specified by buffer into the address space of the current CUDA context and returns in *dptr and *size the base pointer and size of the resulting mapping.

There must be a valid OpenGL context bound to the current thread when this function is called. This must be the same context, or a member of the same shareGroup, as the context that was bound when the buffer was registered.

All streams in the current CUDA context are synchronized with the current GL context.

Parameters:

```
dptr - Returned mapped base pointer
```

size - Returned size of mapping

buffer - The name of the buffer object to map

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_MAP_FAILED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources

4.33.2.3 cuGLMapBufferObjectAsync (CUdeviceptr * dptr, unsigned int * size, GLuint buffer, CUstream hStream)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the buffer object specified by buffer into the address space of the current CUDA context and returns in *dptr and *size the base pointer and size of the resulting mapping.

There must be a valid OpenGL context bound to the current thread when this function is called. This must be the same context, or a member of the same shareGroup, as the context that was bound when the buffer was registered.

Stream hStream in the current CUDA context is synchronized with the current GL context.

Parameters:

dptr - Returned mapped base pointer

size - Returned size of mapping

buffer - The name of the buffer object to map

hStream - Stream to synchronize

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_MAP_FAILED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources

4.33.2.4 cuGLRegisterBufferObject (GLuint buffer)

Deprecated

This function is deprecated as of Cuda 3.0.

Registers the buffer object specified by buffer for access by CUDA. This function must be called before CUDA can map the buffer object. There must be a valid OpenGL context bound to the current thread when this function is called, and the buffer name is resolved by that context.

Parameters:

buffer - The name of the buffer object to register.

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_ALREADY_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsGLRegisterBuffer

4.33.2.5 cuGLSetBufferObjectMapFlags (GLuint buffer, unsigned int Flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Sets the map flags for the buffer object specified by buffer.

Changes to Flags will take effect the next time buffer is mapped. The Flags argument may be any of the following:

CU_GL_MAP_RESOURCE_FLAGS_NONE: Specifies no hints about how this resource will be used. It is
therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default
value.

- CU_GL_MAP_RESOURCE_FLAGS_READ_ONLY: Specifies that CUDA kernels which access this resource will not write to this resource.
- CU_GL_MAP_RESOURCE_FLAGS_WRITE_DISCARD: Specifies that CUDA kernels which access this resource will not read from this resource and will write over the entire contents of the resource, so none of the data previously stored in the resource will be preserved.

If buffer has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned. If buffer is presently mapped for access by CUDA, then CUDA_ERROR_ALREADY_MAPPED is returned.

There must be a valid OpenGL context bound to the current thread when this function is called. This must be the same context, or a member of the same shareGroup, as the context that was bound when the buffer was registered.

Parameters:

```
buffer - Buffer object to unmapFlags - Map flags
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED, CUDA_ERROR_INVALID_CONTEXT,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceSetMapFlags

4.33.2.6 cuGLUnmapBufferObject (GLuint buffer)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the buffer object specified by buffer for access by CUDA.

There must be a valid OpenGL context bound to the current thread when this function is called. This must be the same context, or a member of the same shareGroup, as the context that was bound when the buffer was registered.

All streams in the current CUDA context are synchronized with the current GL context.

Parameters:

```
buffer - Buffer object to unmap
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsUnmapResources

4.33.2.7 cuGLUnmapBufferObjectAsync (GLuint buffer, CUstream hStream)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the buffer object specified by buffer for access by CUDA.

There must be a valid OpenGL context bound to the current thread when this function is called. This must be the same context, or a member of the same shareGroup, as the context that was bound when the buffer was registered.

Stream hStream in the current CUDA context is synchronized with the current GL context.

Parameters:

buffer - Name of the buffer object to unmaphStream - Stream to synchronize

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsUnmapResources

4.33.2.8 cuGLUnregisterBufferObject (GLuint buffer)

Deprecated

This function is deprecated as of Cuda 3.0.

Unregisters the buffer object specified by buffer. This releases any resources associated with the registered buffer. After this call, the buffer may no longer be mapped for access by CUDA.

There must be a valid OpenGL context bound to the current thread when this function is called. This must be the same context, or a member of the same shareGroup, as the context that was bound when the buffer was registered.

Parameters:

buffer - Name of the buffer object to unregister

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cu Graphics Unregister Resource

4.34 Direct3D 9 Interoperability

Modules

• Direct3D 9 Interoperability [DEPRECATED]

Functions

• CUresult cuD3D9CtxCreate (CUcontext *pCtx, CUdevice *pCudaDevice, unsigned int Flags, IDirect3DDevice9 *pD3DDevice)

Create a CUDA context for interoperability with Direct3D 9.

• CUresult cuD3D9GetDevice (CUdevice *pCudaDevice, const char *pszAdapterName)

Gets the CUDA device corresponding to a display adapter.

CUresult cuGraphicsD3D9RegisterResource (CUgraphicsResource *pCudaResource, IDirect3DResource9 *pD3DResource, unsigned int Flags)

Register a Direct3D 9 resource for access by CUDA.

4.34.1 Detailed Description

This section describes the Direct3D 9 interoperability functions of the low-level CUDA driver application programming interface.

4.34.2 Function Documentation

4.34.2.1 cuD3D9CtxCreate (CUcontext * pCtx, CUdevice * pCudaDevice, unsigned int Flags, IDirect3DDevice9 * pD3DDevice)

Creates a new CUDA context, enables interoperability for that context with the Direct3D device pD3DDevice, and associates the created CUDA context with the calling thread. The created CUcontext will be returned in *pCtx. Direct3D resources from this device may be registered and mapped through the lifetime of this CUDA context. If pCudaDevice is non-NULL then the CUdevice on which this CUDA context was created will be returned in *pCudaDevice.

On success, this call will increase the internal reference count on pD3DDevice. This reference count will be decremented upon destruction of this context through cuCtxDestroy(). This context will cease to function if pD3DDevice is destroyed or encounters an error.

Parameters:

```
    pCtx - Returned newly created CUDA context
    pCudaDevice - Returned pointer to the device on which the context was created
    Flags - Context creation flags (see cuCtxCreate() for details)
    pD3DDevice - Direct3D device to create interoperability context with
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D9GetDevice, cuGraphicsD3D9RegisterResource

4.34.2.2 cuD3D9GetDevice (CUdevice * pCudaDevice, const char * pszAdapterName)

Returns in *pCudaDevice the CUDA-compatible device corresponding to the adapter name pszAdapterName obtained from EnumDisplayDevices() or IDirect3D9::GetAdapterIdentifier().

If no device on the adapter with name pszAdapterName is CUDA-compatible, then the call will fail.

Parameters:

```
pCudaDevice - Returned CUDA device corresponding to pszAdapterNamepszAdapterName - Adapter name to query for device
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR INVALID VALUE, CUDA ERROR UNKNOWN
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D9CtxCreate

4.34.2.3 cuGraphicsD3D9RegisterResource (CUgraphicsResource * pCudaResource, IDirect3DResource9 * pD3DResource, unsigned int Flags)

Registers the Direct3D 9 resource pD3DResource for access by CUDA and returns a CUDA handle to pD3Dresource in pCudaResource. The handle returned in pCudaResource may be used to map and unmap this resource until it is unregistered. On success this call will increase the internal reference count on pD3DResource. This reference count will be decremented when this resource is unregistered through cuGraphicsUnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pD3DResource must be one of the following.

- IDirect3DVertexBuffer9: may be accessed through a device pointer
- IDirect3DIndexBuffer9: may be accessed through a device pointer
- IDirect3DSurface9: may be accessed through an array. Only stand-alone objects of type IDirect3DSurface9 may be explicitly shared. In particular, individual mipmap levels and faces of cube maps may not be registered directly. To access individual surfaces associated with a texture, one must register the base texture object.
- IDirect3DBaseTexture9: individual surfaces on this texture may be accessed through an array.

The Flags argument may be used to specify additional parameters at register time. The only valid value for this parameter is

• CU_GRAPHICS_REGISTER_FLAGS_NONE

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized for this context using cuD3D9CtxCreate then CUDA_ERROR_INVALID_CONTEXT is returned. If pD3DResource is of incorrect type or is already registered then CUDA_ERROR_INVALID_HANDLE is returned. If pD3DResource cannot be registered then CUDA_ERROR_UNKNOWN is returned. If Flags is not one of the above specified value then CUDA_ERROR_INVALID_VALUE is returned.

Parameters:

```
    pCudaResource - Returned graphics resource handle
    pD3DResource - Direct3D resource to register
    Flags - Parameters for resource registration
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D9CtxCreate, cuGraphicsUnregisterResource, cuGraphicsMapResources, cuGraphicsSubResourceGetMappedArray, cuGraphicsResourceGetMappedPointer

4.35 Direct3D 9 Interoperability [DEPRECATED]

Functions

CUresult cuD3D9GetDirect3DDevice (IDirect3DDevice9 **ppD3DDevice)

Get the Direct3D 9 device against which the current CUDA context was created.

• CUresult cuD3D9MapResources (unsigned int count, IDirect3DResource9 **ppResource)

Map Direct3D resources for access by CUDA.

• CUresult cuD3D9RegisterResource (IDirect3DResource9 *pResource, unsigned int Flags)

Register a Direct3D resource for access by CUDA.

 CUresult cuD3D9ResourceGetMappedArray (CUarray *pArray, IDirect3DResource9 *pResource, unsigned int Face, unsigned int Level)

Get an array through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

• CUresult cuD3D9ResourceGetMappedPitch (unsigned int *pPitch, unsigned int *pPitchSlice, IDirect3DResource9 *pResource, unsigned int Face, unsigned int Level)

Get the pitch of a subresource of a Direct3D resource which has been mapped for access by CUDA.

CUresult cuD3D9ResourceGetMappedPointer (CUdeviceptr *pDevPtr, IDirect3DResource9 *pResource, unsigned int Face, unsigned int Level)

Get the pointer through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

• CUresult cuD3D9ResourceGetMappedSize (unsigned int *pSize, IDirect3DResource9 *pResource, unsigned int Face, unsigned int Level)

Get the size of a subresource of a Direct3D resource which has been mapped for access by CUDA.

• CUresult cuD3D9ResourceGetSurfaceDimensions (unsigned int *pWidth, unsigned int *pHeight, unsigned int *pDepth, IDirect3DResource9 *pResource, unsigned int Face, unsigned int Level)

Get the dimensions of a registered surface.

CUresult cuD3D9ResourceSetMapFlags (IDirect3DResource9 *pResource, unsigned int Flags)

Set usage flags for mapping a Direct3D resource.

CUresult cuD3D9UnmapResources (unsigned int count, IDirect3DResource9 **ppResource)

Unmaps Direct3D resources.

• CUresult cuD3D9UnregisterResource (IDirect3DResource9 *pResource)

Unregister a Direct3D resource.

4.35.1 Detailed Description

This section describes deprecated Direct3D 9 interoperability functionality.

4.35.2 Function Documentation

4.35.2.1 cuD3D9GetDirect3DDevice (IDirect3DDevice9 ** ppD3DDevice)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *ppD3DDevice the Direct3D device against which this CUDA context was created in cuD3D9CtxCreate().

Parameters:

ppD3DDevice - Returned Direct3D device corresponding to CUDA context

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D9GetDevice

4.35.2.2 cuD3D9MapResources (unsigned int count, IDirect3DResource9 ** ppResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the count Direct3D resources in ppResource for access by CUDA.

The resources in ppResource may be accessed in CUDA kernels until they are unmapped. Direct3D should not access any resources while they are mapped by CUDA. If an application does so the results are undefined.

This function provides the synchronization guarantee that any Direct3D calls issued before cuD3D9MapResources() will complete before any CUDA kernels issued after cuD3D9MapResources() begin.

If any of ppResource have not been registered for use with CUDA or if ppResource contains any duplicate entries, then CUDA_ERROR_INVALID_HANDLE is returned. If any of ppResource are presently mapped for access by CUDA, then CUDA_ERROR_ALREADY_MAPPED is returned.

Parameters:

```
count - Number of resources in ppResourceppResource - Resources to map for CUDA usage
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources

4.35.2.3 cuD3D9RegisterResource (IDirect3DResource9 * pResource, unsigned int Flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Registers the Direct3D resource pResource for access by CUDA.

If this call is successful, then the application will be able to map and unmap this resource until it is unregistered through cuD3D9UnregisterResource(). Also on success, this call will increase the internal reference count on pResource. This reference count will be decremented when this resource is unregistered through cuD3D9UnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pResource must be one of the following.

- IDirect3DVertexBuffer9: Cannot be used with Flags set to CU_D3D9_REGISTER_FLAGS_ARRAY.
- IDirect3DIndexBuffer9: Cannot be used with Flags set to CU_D3D9_REGISTER_FLAGS_ARRAY.
- IDirect3DSurface9: Only stand-alone objects of type IDirect3DSurface9 may be explicitly shared. In particular, individual mipmap levels and faces of cube maps may not be registered directly. To access individual surfaces associated with a texture, one must register the base texture object. For restrictions on the Flags parameter, see type IDirect3DBaseTexture9.
- IDirect3DBaseTexture9: When a texture is registered, all surfaces associated with the all mipmap levels of all faces of the texture will be accessible to CUDA.

The Flags argument specifies the mechanism through which CUDA will access the Direct3D resource. The following values are allowed.

- CU_D3D9_REGISTER_FLAGS_NONE: Specifies that CUDA will access this resource through a CUd-eviceptr. The pointer, size, and (for textures), pitch for each subresource of this allocation may be queried through cuD3D9ResourceGetMappedPointer(), cuD3D9ResourceGetMappedSize(), and cuD3D9ResourceGetMappedPitch() respectively. This option is valid for all resource types.
- CU_D3D9_REGISTER_FLAGS_ARRAY: Specifies that CUDA will access this resource through a CUarray queried on a sub-resource basis through cuD3D9ResourceGetMappedArray(). This option is only valid for resources of type IDirect3DSurface9 and subtypes of IDirect3DBaseTexture9.

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Any resources allocated in D3DPOOL_SYSTEMMEM or D3DPOOL_MANAGED may not be registered with CUDA.

- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized on this context, then CUDA_ERROR_INVALID_CONTEXT is returned. If pResource is of incorrect type (e.g. is a non-stand-alone IDirect3DSurface9) or is already registered, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource cannot be registered then CUDA_ERROR_UNKNOWN is returned.

Parameters:

```
pResource - Resource to register for CUDA accessFlags - Flags for resource registration
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsD3D9RegisterResource

4.35.2.4 cuD3D9ResourceGetMappedArray (CUarray * pArray, IDirect3DResource9 * pResource, unsigned int Face, unsigned int Level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pArray an array through which the subresource of the mapped Direct3D resource pResource which corresponds to Face and Level may be accessed. The value set in pArray may change every time that pResource is mapped.

If pResource is not registered then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D9_REGISTER_FLAGS_ARRAY then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped then CUDA_ERROR_NOT_MAPPED is returned.

For usage requirements of Face and Level parameters, see cuD3D9ResourceGetMappedPointer().

Parameters:

```
    pArray - Returned array corresponding to subresource
    pResource - Mapped resource to access
    Face - Face of resource to access
    Level - Level of resource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsSubResourceGetMappedArray

4.35.2.5 cuD3D9ResourceGetMappedPitch (unsigned int * pPitch, unsigned int * pPitchSlice, IDirect3DResource9 * pResource, unsigned int Face, unsigned int Level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pPitch and *pPitchSlice the pitch and Z-slice pitch of the subresource of the mapped Direct3D resource pResource, which corresponds to Face and Level. The values set in pPitch and pPitchSlice may change every time that pResource is mapped.

The pitch and Z-slice pitch values may be used to compute the location of a sample on a surface as follows.

For a 2D surface, the byte offset of the sample at position x, y from the base pointer of the surface is:

```
y * pitch + (bytes per pixel) * x
```

For a 3D surface, the byte offset of the sample at position x, y, z from the base pointer of the surface is:

```
z* slicePitch + y * pitch + (bytes per pixel) * x
```

Both parameters pPitch and pPitchSlice are optional and may be set to NULL.

If pResource is not of type IDirect3DBaseTexture9 or one of its sub-types or if pResource has not been registered for use with CUDA, then cudaErrorInvalidResourceHandle is returned. If pResource was not registered with usage flags CU_D3D9_REGISTER_FLAGS_NONE, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped for access by CUDA then CUDA_ERROR_NOT_MAPPED is returned.

For usage requirements of Face and Level parameters, see cuD3D9ResourceGetMappedPointer().

Parameters:

```
    pPitch - Returned pitch of subresource
    pPitchSlice - Returned Z-slice pitch of subresource
    pResource - Mapped resource to access
    Face - Face of resource to access
    Level - Level of resource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsSubResourceGetMappedArray

4.35.2.6 cuD3D9ResourceGetMappedPointer (CUdeviceptr * pDevPtr, IDirect3DResource9 * pResource, unsigned int Face, unsigned int Level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pDevPtr the base pointer of the subresource of the mapped Direct3D resource pResource, which corresponds to Face and Level. The value set in pDevPtr may change every time that pResource is mapped.

If pResource is not registered, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D9_REGISTER_FLAGS_NONE, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped, then CUDA_ERROR_NOT_MAPPED is returned.

If pResource is of type IDirect3DCubeTexture9, then Face must one of the values enumerated by type D3DCUBEMAP_FACES. For all other types Face must be 0. If Face is invalid, then CUDA_ERROR_INVALID_-VALUE is returned.

If pResource is of type IDirect3DBaseTexture9, then Level must correspond to a valid mipmap level. At present only mipmap level 0 is supported. For all other types Level must be 0. If Level is invalid, then CUDA_ERROR_-INVALID_VALUE is returned.

Parameters:

pDevPtr - Returned pointer corresponding to subresource

pResource - Mapped resource to access

Face - Face of resource to access

Level - Level of resource to access

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cu Graphics Resource Get Mapped Pointer

4.35.2.7 cuD3D9ResourceGetMappedSize (unsigned int * pSize, IDirect3DResource9 * pResource, unsigned int Face, unsigned int Level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pSize the size of the subresource of the mapped Direct3D resource pResource, which corresponds to Face and Level. The value set in pSize may change every time that pResource is mapped.

If pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D9_REGISTER_FLAGS_NONE, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped for access by CUDA, then CUDA_ERROR_NOT_MAPPED is returned.

For usage requirements of Face and Level parameters, see cuD3D9ResourceGetMappedPointer.

Parameters:

```
    pSize - Returned size of subresource
    pResource - Mapped resource to access
    Face - Face of resource to access
    Level - Level of resource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceGetMappedPointer

4.35.2.8 cuD3D9ResourceGetSurfaceDimensions (unsigned int * pWidth, unsigned int * pHeight, unsigned int * pDepth, IDirect3DResource9 * pResource, unsigned int Face, unsigned int Level)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pWidth, *pHeight, and *pDepth the dimensions of the subresource of the mapped Direct3D resource pResource, which corresponds to Face and Level.

Because anti-aliased surfaces may have multiple samples per pixel, it is possible that the dimensions of a resource will be an integer factor larger than the dimensions reported by the Direct3D runtime.

The parameters pWidth, pHeight, and pDepth are optional. For 2D surfaces, the value returned in *pDepth will be 0.

If pResource is not of type IDirect3DBaseTexture9 or IDirect3DSurface9 or if pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned.

For usage requirements of Face and Level parameters, see cuD3D9ResourceGetMappedPointer().

Parameters:

```
    pWidth - Returned width of surface
    pHeight - Returned height of surface
    pDepth - Returned depth of surface
    pResource - Registered resource to access
    Face - Face of resource to access
    Level - Level of resource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsSubResourceGetMappedArray

4.35.2.9 cuD3D9ResourceSetMapFlags (IDirect3DResource9 * pResource, unsigned int Flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Set Flags for mapping the Direct3D resource pResource.

Changes to Flags will take effect the next time pResource is mapped. The Flags argument may be any of the following:

- CU_D3D9_MAPRESOURCE_FLAGS_NONE: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default value.
- CU_D3D9_MAPRESOURCE_FLAGS_READONLY: Specifies that CUDA kernels which access this resource
 will not write to this resource.
- CU_D3D9_MAPRESOURCE_FLAGS_WRITEDISCARD: Specifies that CUDA kernels which access this resource will not read from this resource and will write over the entire contents of the resource, so none of the data previously stored in the resource will be preserved.

If pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is presently mapped for access by CUDA, then CUDA_ERROR_ALREADY_MAPPED is returned.

Parameters:

pResource - Registered resource to set flags for

Flags - Parameters for resource mapping

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceSetMapFlags

4.35.2.10 cuD3D9UnmapResources (unsigned int count, IDirect3DResource9 ** ppResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the count Direct3D resources in ppResource.

This function provides the synchronization guarantee that any CUDA kernels issued before cuD3D9UnmapResources() will complete before any Direct3D calls issued after cuD3D9UnmapResources() begin.

If any of ppResource have not been registered for use with CUDA or if ppResource contains any duplicate entries, then CUDA_ERROR_INVALID_HANDLE is returned. If any of ppResource are not presently mapped for access by CUDA, then CUDA_ERROR_NOT_MAPPED is returned.

Parameters:

```
count - Number of resources to unmap for CUDAppResource - Resources to unmap for CUDA
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsUnmapResources

4.35.2.11 cuD3D9UnregisterResource (IDirect3DResource9 * pResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Unregisters the Direct3D resource pResource so it is not accessible by CUDA unless registered again.

If pResource is not registered, then CUDA_ERROR_INVALID_HANDLE is returned.

Parameters:

```
pResource - Resource to unregister
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsUnregisterResource

4.36 Direct3D 10 Interoperability

Modules

• Direct3D 10 Interoperability [DEPRECATED]

Functions

 CUresult cuD3D10CtxCreate (CUcontext *pCtx, CUdevice *pCudaDevice, unsigned int Flags, ID3D10Device *pD3DDevice)

Create a CUDA context for interoperability with Direct3D 10.

CUresult cuD3D10GetDevice (CUdevice *pCudaDevice, IDXGIAdapter *pAdapter)
 Gets the CUDA device corresponding to a display adapter.

• CUresult cuGraphicsD3D10RegisterResource (CUgraphicsResource *pCudaResource, ID3D10Resource *pD3DResource, unsigned int Flags)

Register a Direct3D 10 resource for access by CUDA.

4.36.1 Detailed Description

This section describes the Direct3D 10 interoperability functions of the low-level CUDA driver application programming interface.

4.36.2 Function Documentation

4.36.2.1 cuD3D10CtxCreate (CUcontext * pCtx, CUdevice * pCudaDevice, unsigned int Flags, ID3D10Device * pD3DDevice)

Creates a new CUDA context, enables interoperability for that context with the Direct3D device pD3DDevice, and associates the created CUDA context with the calling thread. The created CUcontext will be returned in *pCtx. Direct3D resources from this device may be registered and mapped through the lifetime of this CUDA context. If pCudaDevice is non-NULL then the CUdevice on which this CUDA context was created will be returned in *pCudaDevice.

On success, this call will increase the internal reference count on pD3DDevice. This reference count will be decremented upon destruction of this context through cuCtxDestroy(). This context will cease to function if pD3DDevice is destroyed or encounters an error.

Parameters:

```
    pCtx - Returned newly created CUDA context
    pCudaDevice - Returned pointer to the device on which the context was created
    Flags - Context creation flags (see cuCtxCreate() for details)
    pD3DDevice - Direct3D device to create interoperability context with
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D10GetDevice, cuGraphicsD3D10RegisterResource

4.36.2.2 cuD3D10GetDevice (CUdevice * pCudaDevice, IDXGIAdapter * pAdapter)

Returns in *pCudaDevice the CUDA-compatible device corresponding to the adapter pAdapter obtained from IDXGIFactory::EnumAdapters.

If no device on pAdapter is CUDA-compatible then the call will fail.

Parameters:

```
\begin{tabular}{ll} $\it pCudaDevice$ - Returned CUDA device corresponding to pAdapter \\ $\it pAdapter$ - Adapter to query for CUDA device \\ \end{tabular}
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_UNKNOWN
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D10CtxCreate

4.36.2.3 cuGraphicsD3D10RegisterResource (CUgraphicsResource * pCudaResource, ID3D10Resource * pD3DResource, unsigned int Flags)

Registers the Direct3D 10 resource pD3DResource for access by CUDA and returns a CUDA handle to pD3Dresource in pCudaResource. The handle returned in pCudaResource may be used to map and unmap this resource until it is unregistered. On success this call will increase the internal reference count on pD3DResource. This reference count will be decremented when this resource is unregistered through cuGraphicsUnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pD3DResource must be one of the following.

- ID3D10Buffer: may be accessed through a device pointer.
- ID3D10Texture1D: individual subresources of the texture may be accessed via arrays
- ID3D10Texture2D: individual subresources of the texture may be accessed via arrays
- ID3D10Texture3D: individual subresources of the texture may be accessed via arrays

The Flags argument may be used to specify additional parameters at register time. The only valid value for this parameter is

CU_GRAPHICS_REGISTER_FLAGS_NONE

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized for this context using cuD3D10CtxCreate then CUDA_ERROR_INVALID_CONTEXT is returned. If pD3DResource is of incorrect type or is already registered then CUDA_ERROR_INVALID_HANDLE is returned. If pD3DResource cannot be registered then CUDA_ERROR_UNKNOWN is returned. If Flags is not one of the above specified value then CUDA_ERROR_INVALID_VALUE is returned.

Parameters:

```
    pCudaResource - Returned graphics resource handle
    pD3DResource - Direct3D resource to register
    Flags - Parameters for resource registration
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D10CtxCreate, cuGraphicsUnregisterResource, cuGraphicsMapResources, cuGraphicsSubResourceGetMappedArray, cuGraphicsResourceGetMappedPointer

4.37 Direct3D 10 Interoperability [DEPRECATED]

Functions

CUresult cuD3D10MapResources (unsigned int count, ID3D10Resource **ppResources)
 Map Direct3D resources for access by CUDA.

• CUresult cuD3D10RegisterResource (ID3D10Resource *pResource, unsigned int Flags)

Register a Direct3D resource for access by CUDA.

 CUresult cuD3D10ResourceGetMappedArray (CUarray *pArray, ID3D10Resource *pResource, unsigned int SubResource)

Get an array through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

 CUresult cuD3D10ResourceGetMappedPitch (unsigned int *pPitch, unsigned int *pPitchSlice, ID3D10Resource *pResource, unsigned int SubResource)

Get the pitch of a subresource of a Direct3D resource which has been mapped for access by CUDA.

CUresult cuD3D10ResourceGetMappedPointer (CUdeviceptr *pDevPtr, ID3D10Resource *pResource, unsigned int SubResource)

Get a pointer through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA.

 CUresult cuD3D10ResourceGetMappedSize (unsigned int *pSize, ID3D10Resource *pResource, unsigned int SubResource)

Get the size of a subresource of a Direct3D resource which has been mapped for access by CUDA.

• CUresult cuD3D10ResourceGetSurfaceDimensions (unsigned int *pWidth, unsigned int *pHeight, unsigned int *pDepth, ID3D10Resource *pResource, unsigned int SubResource)

Get the dimensions of a registered surface.

- CUresult cuD3D10ResourceSetMapFlags (ID3D10Resource *pResource, unsigned int Flags)
 Set usage flags for mapping a Direct3D resource.
- CUresult cuD3D10UnmapResources (unsigned int count, ID3D10Resource **ppResources)
 Unmap Direct3D resources.
- CUresult cuD3D10UnregisterResource (ID3D10Resource *pResource)

 Unregister a Direct3D resource.

4.37.1 Detailed Description

This section describes deprecated Direct3D 10 interoperability functionality.

4.37.2 Function Documentation

4.37.2.1 cuD3D10MapResources (unsigned int count, ID3D10Resource ** ppResources)

Deprecated

This function is deprecated as of Cuda 3.0.

Maps the count Direct3D resources in ppResources for access by CUDA.

The resources in ppResources may be accessed in CUDA kernels until they are unmapped. Direct3D should not access any resources while they are mapped by CUDA. If an application does so, the results are undefined.

This function provides the synchronization guarantee that any Direct3D calls issued before cuD3D10MapResources() will complete before any CUDA kernels issued after cuD3D10MapResources() begin.

If any of ppResources have not been registered for use with CUDA or if ppResources contains any duplicate entries, then CUDA_ERROR_INVALID_HANDLE is returned. If any of ppResources are presently mapped for access by CUDA, then CUDA_ERROR_ALREADY_MAPPED is returned.

Parameters:

```
count - Number of resources to map for CUDAppResources - Resources to map for CUDA
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources

4.37.2.2 cuD3D10RegisterResource (ID3D10Resource * pResource, unsigned int Flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Registers the Direct3D resource pResource for access by CUDA.

If this call is successful, then the application will be able to map and unmap this resource until it is unregistered through cuD3D10UnregisterResource(). Also on success, this call will increase the internal reference count on pResource. This reference count will be decremented when this resource is unregistered through cuD3D10UnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pResource must be one of the following.

- ID3D10Buffer: Cannot be used with Flags set to CU_D3D10_REGISTER_FLAGS_ARRAY.
- ID3D10Texture1D: No restrictions.
- ID3D10Texture2D: No restrictions.
- ID3D10Texture3D: No restrictions.

The Flags argument specifies the mechanism through which CUDA will access the Direct3D resource. The following values are allowed.

• CU_D3D10_REGISTER_FLAGS_NONE: Specifies that CUDA will access this resource through a CUdeviceptr. The pointer, size, and (for textures), pitch for each subresource of this allocation may be queried through cuD3D10ResourceGetMappedPointer(), cuD3D10ResourceGetMappedSize(), and cuD3D10ResourceGetMappedPitch() respectively. This option is valid for all resource types.

CU_D3D10_REGISTER_FLAGS_ARRAY: Specifies that CUDA will access this resource through a CUarray queried on a sub-resource basis through cuD3D10ResourceGetMappedArray(). This option is only valid for resources of type ID3D10Texture1D, ID3D10Texture2D, and ID3D10Texture3D.

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- · Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized on this context then CUDA_ERROR_INVALID_CONTEXT is returned. If pResource is of incorrect type or is already registered, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource cannot be registered, then CUDA_ERROR_UNKNOWN is returned.

Parameters:

pResource - Resource to registerFlags - Parameters for resource registration

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsD3D10RegisterResource

4.37.2.3 cuD3D10ResourceGetMappedArray (CUarray * pArray, ID3D10Resource * pResource, unsigned int SubResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pArray an array through which the subresource of the mapped Direct3D resource pResource, which corresponds to SubResource may be accessed. The value set in pArray may change every time that pResource is mapped.

If pResource is not registered, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D10_REGISTER_FLAGS_ARRAY, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped, then CUDA_ERROR_NOT_MAPPED is returned.

For usage requirements of the SubResource parameter, see cuD3D10ResourceGetMappedPointer().

Parameters:

pArray - Returned array corresponding to subresource
 pResource - Mapped resource to access
 SubResource - Subresource of pResource to access

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsSubResourceGetMappedArray

4.37.2.4 cuD3D10ResourceGetMappedPitch (unsigned int * pPitch, unsigned int * pPitchSlice, ID3D10Resource * pResource, unsigned int SubResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pPitch and *pPitchSlice the pitch and Z-slice pitch of the subresource of the mapped Direct3D resource pResource, which corresponds to SubResource. The values set in pPitch and pPitchSlice may change every time that pResource is mapped.

The pitch and Z-slice pitch values may be used to compute the location of a sample on a surface as follows.

For a 2D surface, the byte offset of the sample at position x, y from the base pointer of the surface is:

```
y * pitch + (bytes per pixel) * x
```

For a 3D surface, the byte offset of the sample at position x, y, z from the base pointer of the surface is:

```
z* slicePitch + y* pitch + (bytes per pixel) * x
```

Both parameters pPitch and pPitchSlice are optional and may be set to NULL.

If pResource is not of type IDirect3DBaseTexture10 or one of its sub-types or if pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D10_REGISTER_FLAGS_NONE, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped for access by CUDA, then CUDA_ERROR_NOT_MAPPED is returned.

For usage requirements of the SubResource parameter, see cuD3D10ResourceGetMappedPointer().

Parameters:

pPitch - Returned pitch of subresource

```
pPitchSlice - Returned Z-slice pitch of subresourcepResource - Mapped resource to accessSubResource - Subresource of pResource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsSubResourceGetMappedArray

4.37.2.5 cuD3D10ResourceGetMappedPointer (CUdeviceptr * pDevPtr, ID3D10Resource * pResource, unsigned int SubResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pDevPtr the base pointer of the subresource of the mapped Direct3D resource pResource, which corresponds to SubResource. The value set in pDevPtr may change every time that pResource is mapped.

If pResource is not registered, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D10_REGISTER_FLAGS_NONE, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped, then CUDA_ERROR_NOT_MAPPED is returned.

If pResource is of type ID3D10Buffer, then SubResource must be 0. If pResource is of any other type, then the value of SubResource must come from the subresource calculation in D3D10CalcSubResource().

Parameters:

```
pDevPtr - Returned pointer corresponding to subresourcepResource - Mapped resource to accessSubResource - Subresource of pResource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceGetMappedPointer

4.37.2.6 cuD3D10ResourceGetMappedSize (unsigned int * pSize, ID3D10Resource * pResource, unsigned int SubResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pSize the size of the subresource of the mapped Direct3D resource pResource, which corresponds to SubResource. The value set in pSize may change every time that pResource is mapped.

If pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource was not registered with usage flags CU_D3D10_REGISTER_FLAGS_NONE, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is not mapped for access by CUDA, then CUDA_ERROR_NOT_MAPPED is returned.

For usage requirements of the SubResource parameter, see cuD3D10ResourceGetMappedPointer().

Parameters:

```
pSize - Returned size of subresourcepResource - Mapped resource to accessSubResource - Subresource of pResource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceGetMappedPointer

4.37.2.7 cuD3D10ResourceGetSurfaceDimensions (unsigned int * pWidth, unsigned int * pHeight, unsigned int * pDepth, ID3D10Resource * pResource, unsigned int SubResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Returns in *pWidth, *pHeight, and *pDepth the dimensions of the subresource of the mapped Direct3D resource pResource, which corresponds to SubResource.

Because anti-aliased surfaces may have multiple samples per pixel, it is possible that the dimensions of a resource will be an integer factor larger than the dimensions reported by the Direct3D runtime.

The parameters pWidth, pHeight, and pDepth are optional. For 2D surfaces, the value returned in *pDepth will be 0.

If pResource is not of type IDirect3DBaseTexture10 or IDirect3DSurface10 or if pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned.

For usage requirements of the SubResource parameter, see cuD3D10ResourceGetMappedPointer().

Parameters:

```
    pWidth - Returned width of surface
    pHeight - Returned height of surface
    pDepth - Returned depth of surface
    pResource - Registered resource to access
    SubResource - Subresource of pResource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsSubResourceGetMappedArray

4.37.2.8 cuD3D10ResourceSetMapFlags (ID3D10Resource * pResource, unsigned int Flags)

Deprecated

This function is deprecated as of Cuda 3.0.

Set flags for mapping the Direct3D resource pResource.

Changes to flags will take effect the next time pResource is mapped. The Flags argument may be any of the following.

- CU_D3D10_MAPRESOURCE_FLAGS_NONE: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default value.
- CU_D3D10_MAPRESOURCE_FLAGS_READONLY: Specifies that CUDA kernels which access this resource will not write to this resource.
- CU_D3D10_MAPRESOURCE_FLAGS_WRITEDISCARD: Specifies that CUDA kernels which access this
 resource will not read from this resource and will write over the entire contents of the resource, so none of the
 data previously stored in the resource will be preserved.

If pResource has not been registered for use with CUDA, then CUDA_ERROR_INVALID_HANDLE is returned. If pResource is presently mapped for access by CUDA then CUDA_ERROR_ALREADY_MAPPED is returned.

Parameters:

```
pResource - Registered resource to set flags forFlags - Parameters for resource mapping
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceSetMapFlags

4.37.2.9 cuD3D10UnmapResources (unsigned int count, ID3D10Resource ** ppResources)

Deprecated

This function is deprecated as of Cuda 3.0.

Unmaps the count Direct3D resources in ppResources.

This function provides the synchronization guarantee that any CUDA kernels issued before cuD3D10UnmapResources() will complete before any Direct3D calls issued after cuD3D10UnmapResources() begin.

If any of ppResources have not been registered for use with CUDA or if ppResources contains any duplicate entries, then CUDA_ERROR_INVALID_HANDLE is returned. If any of ppResources are not presently mapped for access by CUDA, then CUDA_ERROR_NOT_MAPPED is returned.

Parameters:

```
count - Number of resources to unmap for CUDAppResources - Resources to unmap for CUDA
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsUnmapResources

4.37.2.10 cuD3D10UnregisterResource (ID3D10Resource * pResource)

Deprecated

This function is deprecated as of Cuda 3.0.

Unregisters the Direct3D resource pResource so it is not accessible by CUDA unless registered again.

If pResource is not registered, then CUDA_ERROR_INVALID_HANDLE is returned.

Parameters:

pResource - Resources to unregister

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cu Graphics Unregister Resource

4.38 Direct3D 11 Interoperability

Functions

 CUresult cuD3D11CtxCreate (CUcontext *pCtx, CUdevice *pCudaDevice, unsigned int Flags, ID3D11Device *pD3DDevice)

Create a CUDA context for interoperability with Direct3D 11.

• CUresult cuD3D11GetDevice (CUdevice *pCudaDevice, IDXGIAdapter *pAdapter)

Gets the CUDA device corresponding to a display adapter.

CUresult cuGraphicsD3D11RegisterResource (CUgraphicsResource *pCudaResource, ID3D11Resource *pD3DResource, unsigned int Flags)

Register a Direct3D 11 resource for access by CUDA.

4.38.1 Detailed Description

This section describes the Direct3D 11 interoperability functions of the low-level CUDA driver application programming interface.

4.38.2 Function Documentation

4.38.2.1 cuD3D11CtxCreate (CUcontext * pCtx, CUdevice * pCudaDevice, unsigned int Flags, ID3D11Device * pD3DDevice)

Creates a new CUDA context, enables interoperability for that context with the Direct3D device pD3DDevice, and associates the created CUDA context with the calling thread. The created CUcontext will be returned in *pCtx. Direct3D resources from this device may be registered and mapped through the lifetime of this CUDA context. If pCudaDevice is non-NULL then the CUdevice on which this CUDA context was created will be returned in *pCudaDevice.

On success, this call will increase the internal reference count on pD3DDevice. This reference count will be decremented upon destruction of this context through cuCtxDestroy(). This context will cease to function if pD3DDevice is destroyed or encounters an error.

Parameters:

```
    pCtx - Returned newly created CUDA context
    pCudaDevice - Returned pointer to the device on which the context was created
    Flags - Context creation flags (see cuCtxCreate() for details)
    pD3DDevice - Direct3D device to create interoperability context with
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D11GetDevice, cuGraphicsD3D11RegisterResource

4.38.2.2 cuD3D11GetDevice (CUdevice * pCudaDevice, IDXGIAdapter * pAdapter)

Returns in *pCudaDevice the CUDA-compatible device corresponding to the adapter pAdapter obtained from IDXGIFactory::EnumAdapters.

If no device on pAdapter is CUDA-compatible then the call will fail.

Parameters:

```
pCudaDevice - Returned CUDA device corresponding to pAdapter pAdapter - Adapter to query for CUDA device
```

Returns:

```
CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_UNKNOWN
```

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D11CtxCreate

4.38.2.3 cuGraphicsD3D11RegisterResource (CUgraphicsResource * pCudaResource, ID3D11Resource * pD3DResource, unsigned int Flags)

Registers the Direct3D 11 resource pD3DResource for access by CUDA and returns a CUDA handle to pD3Dresource in pCudaResource. The handle returned in pCudaResource may be used to map and unmap this resource until it is unregistered. On success this call will increase the internal reference count on pD3DResource. This reference count will be decremented when this resource is unregistered through cuGraphicsUnregisterResource().

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of pD3DResource must be one of the following.

- ID3D11Buffer: may be accessed through a device pointer.
- ID3D11Texture1D: individual subresources of the texture may be accessed via arrays
- ID3D11Texture2D: individual subresources of the texture may be accessed via arrays
- ID3D11Texture3D: individual subresources of the texture may be accessed via arrays

The Flags argument may be used to specify additional parameters at register time. The only valid value for this parameter is

• CU_GRAPHICS_REGISTER_FLAGS_NONE

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.
- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized for this context using cuD3D11CtxCreate then CUDA_ERROR_INVALID_CONTEXT is returned. If pD3DResource is of incorrect type or is already registered then CUDA_ERROR_INVALID_HANDLE is returned. If pD3DResource cannot be registered then CUDA_ERROR_UNKNOWN is returned. If Flags is not one of the above specified value then CUDA_ERROR_INVALID_VALUE is returned.

Parameters:

```
    pCudaResource - Returned graphics resource handle
    pD3DResource - Direct3D resource to register
    Flags - Parameters for resource registration
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_OUT_OF_MEMORY, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuD3D11CtxCreate, cuGraphicsUnregisterResource, cuGraphicsMapResources, cuGraphicsSubResourceGetMappedArray, cuGraphicsResourceGetMappedPointer

4.39 Graphics Interoperability

Functions

• CUresult cuGraphicsMapResources (unsigned int count, CUgraphicsResource *resources, CUstream hStream)

Map graphics resources for access by CUDA.

CUresult cuGraphicsResourceGetMappedPointer (CUdeviceptr *pDevPtr, unsigned int *pSize, CUgraphicsResource resource)

Get an device pointer through which to access a mapped graphics resource.

• CUresult cuGraphicsResourceSetMapFlags (CUgraphicsResource resource, unsigned int flags)

Set usage flags for mapping a graphics resource.

• CUresult cuGraphicsSubResourceGetMappedArray (CUarray *pArray, CUgraphicsResource resource, unsigned int arrayIndex, unsigned int mipLevel)

Get an array through which to access a subresource of a mapped graphics resource.

CUresult cuGraphicsUnmapResources (unsigned int count, CUgraphicsResource *resources, CUstream hStream)

Unmap graphics resources.

• CUresult cuGraphicsUnregisterResource (CUgraphicsResource resource)

Unregisters a graphics resource for access by CUDA.

4.39.1 Detailed Description

This section describes the graphics interoperability functions of the low-level CUDA driver application programming interface.

4.39.2 Function Documentation

4.39.2.1 cuGraphicsMapResources (unsigned int *count*, CUgraphicsResource * *resources*, CUstream *hStream*)

Maps the count graphics resources in resources for access by CUDA.

The resources in resources may be accessed by CUDA until they are unmapped. The graphics API from which resources were registered should not access any resources while they are mapped by CUDA. If an application does so, the results are undefined.

This function provides the synchronization guarantee that any graphics calls issued before cuGraphicsMapResources() will complete before any subsequent CUDA work issued in stream begins.

If resources includes any duplicate entries then CUDA_ERROR_INVALID_HANDLE is returned. If any of resources are presently mapped for access by CUDA then CUDA_ERROR_ALREADY_MAPPED is returned.

Parameters:

count - Number of resources to map

resources - Resources to map for CUDA usage **hStream** - Stream with which to synchronize

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

 $cuGraphics Resource Get Mapped Pointer\ cuGraphics SubResource Get Mapped Array\ cuGraphics Unmap Resources$

4.39.2.2 cuGraphicsResourceGetMappedPointer (CUdeviceptr * *pDevPtr*, unsigned int * *pSize*, CUgraphicsResource *resource*)

Returns in *pDevPtr a pointer through which the mapped graphics resource resource may be accessed. Returns in pSize the size of the memory in bytes which may be accessed from that pointer. The value set in pPointer may change every time that resource is mapped.

If resource is not a buffer then it cannot be accessed via a pointer and CUDA_ERROR_NOT_MAPPED_AS_-POINTER is returned. If resource is not mapped then CUDA_ERROR_NOT_MAPPED is returned. *

Parameters:

```
pDevPtr - Returned pointer through which resource may be accessedpSize - Returned size of the buffer accessible starting at *pPointerresource - Mapped resource to access
```

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED CUDA_ERROR_NOT_MAPPED AS_POINTER

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources, cuGraphicsSubResourceGetMappedArray

4.39.2.3 cuGraphicsResourceSetMapFlags (CUgraphicsResource resource, unsigned int flags)

Set flags for mapping the graphics resource resource.

Changes to flags will take effect the next time resource is mapped. The flags argument may be any of the following:

CU_GRAPHICS_MAP_RESOURCE_FLAGS_NONE: Specifies no hints about how this resource will be used.
 It is therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default value.

- CU_GRAPHICS_MAP_RESOURCE_FLAGS_READONLY: Specifies that CUDA kernels which access this
 resource will not write to this resource.
- CU_GRAPHICS_MAP_RESOURCE_FLAGS_WRITEDISCARD: Specifies that CUDA kernels which access
 this resource will not read from this resource and will write over the entire contents of the resource, so none of
 the data previously stored in the resource will be preserved.

If resource is presently mapped for access by CUDA then CUDA_ERROR_ALREADY_MAPPED is returned. If flags is not one of the above values then CUDA_ERROR_INVALID_VALUE is returned.

Parameters:

resource - Registered resource to set flags for

flags - Parameters for resource mapping

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_ALREADY_MAPPED

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources

4.39.2.4 cuGraphicsSubResourceGetMappedArray (CUarray * pArray, CUgraphicsResource resource, unsigned int arrayIndex, unsigned int mipLevel)

Returns in *pArray an array through which the subresource of the mapped graphics resource resource which corresponds to array index arrayIndex and mipmap level mipLevel may be accessed. The value set in *pArray may change every time that resource is mapped.

If resource is not a texture then it cannot be accessed via an array and CUDA_ERROR_NOT_MAPPED_AS_-ARRAY is returned. If arrayIndex is not a valid array index for resource then CUDA_ERROR_INVALID_-VALUE is returned. If mipLevel is not a valid mipmap level for resource then CUDA_ERROR_INVALID_-VALUE is returned. If resource is not mapped then CUDA_ERROR_NOT_MAPPED is returned.

Parameters:

pArray - Returned array through which a subresource of resource may be accessed

resource - Mapped resource to access

arrayIndex - Array index for array textures or cubemap face index as defined by CUarray_cubemap_face for cubemap textures for the subresource to access

mipLevel - Mipmap level for the subresource to access

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_VALUE, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED CUDA_ERROR_NOT_MAPPED_AS_ARRAY

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsResourceGetMappedPointer

4.39.2.5 cuGraphicsUnmapResources (unsigned int *count*, CUgraphicsResource * *resources*, CUstream *hStream*)

Unmaps the count graphics resources in resources.

Once unmapped, the resources in resources may not be accessed by CUDA until they are mapped again.

This function provides the synchronization guarantee that any CUDA work issued in stream before cuGraphicsUnmapResources() will complete before any subsequently issued graphics work begins.

If resources includes any duplicate entries then CUDA_ERROR_INVALID_HANDLE is returned. If any of resources are not presently mapped for access by CUDA then CUDA_ERROR_NOT_MAPPED is returned.

Parameters:

count - Number of resources to unmap

resources - Resources to unmap

hStream - Stream with which to synchronize

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_NOT_MAPPED, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsMapResources

4.39.2.6 cuGraphicsUnregisterResource (CUgraphicsResource resource)

Unregisters the graphics resource resource so it is not accessible by CUDA unless registered again.

If resource is invalid then CUDA_ERROR_INVALID_HANDLE is returned.

Parameters:

resource - Resource to unregister

Returns:

CUDA_SUCCESS, CUDA_ERROR_DEINITIALIZED, CUDA_ERROR_NOT_INITIALIZED, CUDA_ERROR_INVALID_CONTEXT, CUDA_ERROR_INVALID_HANDLE, CUDA_ERROR_UNKNOWN

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuGraphicsD3D9RegisterResource, cuGraphicsD3D10RegisterResource, cuGraphicsD3D11RegisterResource, cuGraphicsGLRegisterBuffer, cuGraphicsGLRegisterImage

4.40 Data types used by CUDA driver

Data Structures

- struct CUDA ARRAY3D DESCRIPTOR
- struct CUDA_ARRAY_DESCRIPTOR
- struct CUDA_MEMCPY2D_st
- struct CUDA_MEMCPY3D_st
- struct CUdevprop_st

Data types used by CUDA driver

Data types used by CUDA driver

Author:

NVIDIA Corporation

```
enum CUaddress_mode_enum {
 CU_TR_ADDRESS_MODE_WRAP,
 CU_TR_ADDRESS_MODE_CLAMP,
 CU_TR_ADDRESS_MODE_MIRROR }
enum CUarray_cubemap_face_enum {
 CU_CUBEMAP_FACE_POSITIVE_X,
 CU_CUBEMAP_FACE_NEGATIVE_X,
 CU_CUBEMAP_FACE_POSITIVE_Y,
 CU_CUBEMAP_FACE_NEGATIVE_Y,
 CU_CUBEMAP_FACE_POSITIVE_Z,
 CU_CUBEMAP_FACE_NEGATIVE_Z }
• enum CUarray_format_enum {
 CU_AD_FORMAT_UNSIGNED_INT8,
 CU_AD_FORMAT_UNSIGNED_INT16,
 CU_AD_FORMAT_UNSIGNED_INT32,
 CU_AD_FORMAT_SIGNED_INT8,
 CU_AD_FORMAT_SIGNED_INT16,
 CU_AD_FORMAT_SIGNED_INT32,
 CU_AD_FORMAT_HALF,
 CU AD FORMAT FLOAT }
• enum CUcomputemode_enum {
 CU_COMPUTEMODE_DEFAULT,
 CU COMPUTEMODE EXCLUSIVE,
 CU_COMPUTEMODE_PROHIBITED }
```

238 Module Documentation

```
enum CUctx_flags_enum {
 CU_CTX_SCHED_AUTO,
 CU_CTX_SCHED_SPIN,
 CU_CTX_SCHED_YIELD,
 CU_CTX_BLOCKING_SYNC,
 CU_CTX_MAP_HOST,
 CU_CTX_LMEM_RESIZE_TO_MAX }
• enum cudaError_enum {
 CUDA SUCCESS,
 CUDA_ERROR_INVALID_VALUE,
 CUDA_ERROR_OUT_OF_MEMORY,
 CUDA_ERROR_NOT_INITIALIZED,
 CUDA_ERROR_DEINITIALIZED,
 CUDA_ERROR_NO_DEVICE,
 CUDA_ERROR_INVALID_DEVICE,
 CUDA_ERROR_INVALID_IMAGE,
 CUDA_ERROR_INVALID_CONTEXT,
 CUDA_ERROR_CONTEXT_ALREADY_CURRENT,
 CUDA_ERROR_MAP_FAILED,
 CUDA_ERROR_UNMAP_FAILED,
 CUDA_ERROR_ARRAY_IS_MAPPED,
 CUDA_ERROR_ALREADY_MAPPED,
 CUDA_ERROR_NO_BINARY_FOR_GPU,
 CUDA_ERROR_ALREADY_ACQUIRED,
 CUDA_ERROR_NOT_MAPPED,
 CUDA_ERROR_NOT_MAPPED_AS_ARRAY,
 CUDA ERROR NOT MAPPED AS POINTER,
 CUDA_ERROR_INVALID_SOURCE,
 CUDA_ERROR_FILE_NOT_FOUND,
 CUDA_ERROR_INVALID_HANDLE,
 CUDA_ERROR_NOT_FOUND,
 CUDA_ERROR_NOT_READY,
 CUDA_ERROR_LAUNCH_FAILED,
 CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES,
 CUDA_ERROR_LAUNCH_TIMEOUT,
 CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING,
 CUDA ERROR POINTER IS 64BIT,
 CUDA_ERROR_SIZE_IS_64BIT,
 CUDA_ERROR_UNKNOWN }
```

```
• enum CUdevice attribute enum {
 CU DEVICE ATTRIBUTE MAX THREADS PER BLOCK,
 CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X,
 CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y,
 CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z,
 CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X,
 CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y,
 CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z,
 CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK,
 CU_DEVICE_ATTRIBUTE_SHARED_MEMORY_PER_BLOCK,
 CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY,
 CU_DEVICE_ATTRIBUTE_WARP_SIZE,
 CU_DEVICE_ATTRIBUTE_MAX_PITCH,
 CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK,
 CU DEVICE ATTRIBUTE REGISTERS PER BLOCK,
 CU DEVICE ATTRIBUTE CLOCK RATE,
 CU DEVICE ATTRIBUTE TEXTURE ALIGNMENT,
 CU_DEVICE_ATTRIBUTE_GPU_OVERLAP,
 CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT,
 CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT,
 CU_DEVICE_ATTRIBUTE_INTEGRATED,
 CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY,
 CU_DEVICE_ATTRIBUTE_COMPUTE_MODE,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_WIDTH,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_WIDTH,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_HEIGHT,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_WIDTH,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_HEIGHT,
 CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_NUMSLICES,
 CU_DEVICE_ATTRIBUTE_SURFACE_ALIGNMENT,
 CU_DEVICE_ATTRIBUTE_CONCURRENT_KERNELS,
 CU_DEVICE_ATTRIBUTE_ECC_ENABLED }
enum CUevent_flags_enum {
 CU_EVENT_DEFAULT,
 CU EVENT BLOCKING SYNC }

    enum CUfilter mode enum {

 CU_TR_FILTER_MODE_POINT,
 CU_TR_FILTER_MODE_LINEAR }

    enum CUfunc_cache_enum
```

240 Module Documentation

```
• enum CUfunction_attribute_enum {
 CU_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK,
 CU_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES,
 CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES,
 CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES,
 CU_FUNC_ATTRIBUTE_NUM_REGS,
 CU_FUNC_ATTRIBUTE_PTX_VERSION,
 CU FUNC ATTRIBUTE BINARY VERSION }
• enum CUgraphicsMapResourceFlags_enum
• enum CUgraphicsRegisterFlags_enum
• enum CUjit_fallback_enum {
 CU_PREFER_PTX,
 CU_PREFER_BINARY }

    enum CUjit_option_enum {

 CU_JIT_MAX_REGISTERS,
 CU_JIT_THREADS_PER_BLOCK,
 CU_JIT_WALL_TIME,
 CU_JIT_INFO_LOG_BUFFER,
 CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES,
 CU_JIT_ERROR_LOG_BUFFER,
 CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES,
 CU_JIT_OPTIMIZATION_LEVEL,
 CU_JIT_TARGET_FROM_CUCONTEXT,
 CU_JIT_TARGET,
 CU JIT FALLBACK STRATEGY }
enum CUjit_target_enum {
 CU_TARGET_COMPUTE_10,
 CU_TARGET_COMPUTE_11,
 CU_TARGET_COMPUTE_12,
 CU_TARGET_COMPUTE_13,
 CU_TARGET_COMPUTE_20 }
• enum CUmemorytype_enum {
 CU MEMORYTYPE HOST,
 CU_MEMORYTYPE_DEVICE,
 CU MEMORYTYPE ARRAY }
• typedef enum CUaddress mode enum CUaddress mode
• typedef struct CUarray_st * CUarray
    CUDA array.
• typedef enum CUarray_cubemap_face_enum CUarray_cubemap_face
• typedef enum CUarray_format_enum CUarray_format
• typedef enum CUcomputemode enum CUcomputemode
• typedef struct CUctx_st * CUcontext
```

CUDA context.

- typedef enum CUctx_flags_enum CUctx_flags
- typedef struct CUDA_MEMCPY2D_st CUDA_MEMCPY2D
- typedef struct CUDA_MEMCPY3D_st CUDA_MEMCPY3D
- typedef int CUdevice

CUDA device.

- typedef enum CUdevice_attribute_enum CUdevice_attribute
- typedef unsigned int CUdeviceptr

CUDA device pointer.

- typedef struct CUdevprop st CUdevprop
- typedef struct CUevent_st * CUevent

CUDA event.

- typedef enum CUevent_flags_enum CUevent_flags
- typedef enum CUfilter_mode_enum CUfilter_mode
- typedef enum CUfunc_cache_enum CUfunc_cache
- typedef struct CUfunc_st * CUfunction

CUDA function.

- typedef enum CUfunction_attribute_enum CUfunction_attribute
- typedef enum CUgraphicsMapResourceFlags_enum CUgraphicsMapResourceFlags
- typedef enum CUgraphicsRegisterFlags enum CUgraphicsRegisterFlags
- typedef struct CUgraphicsResource_st * CUgraphicsResource

CUDA graphics interop resource.

- typedef enum CUjit_fallback_enum CUjit_fallback
- typedef enum CUjit_option_enum CUjit_option
- typedef enum CUjit_target_enum CUjit_target
- typedef enum CUmemorytype_enum CUmemorytype
- typedef struct CUmod_st * CUmodule

CUDA module.

- typedef enum cudaError_enum CUresult
- typedef struct CUstream st * CUstream

CUDA stream.

• typedef struct CUtexref st * CUtexref

CUDA texture reference.

- typedef struct CUuuid st CUuuid
- #define CU_MEMHOSTALLOC_DEVICEMAP
- #define CU_MEMHOSTALLOC_PORTABLE
- #define CU_MEMHOSTALLOC_WRITECOMBINED
- #define CU_PARAM_TR_DEFAULT
- #define CU_TRSA_OVERRIDE_FORMAT
- #define CU_TRSF_NORMALIZED_COORDINATES
- #define CU_TRSF_READ_AS_INTEGER
- #define CUDA_ARRAY3D_2DARRAY
- #define CUDA_VERSION

242 Module Documentation

4.40.1 Define Documentation

4.40.1.1 #define CU_MEMHOSTALLOC_DEVICEMAP

If set, host memory is mapped into CUDA address space and cuMemHostGetDevicePointer() may be called on the host pointer. Flag for cuMemHostAlloc()

4.40.1.2 #define CU_MEMHOSTALLOC_PORTABLE

If set, host memory is portable between CUDA contexts. Flag for cuMemHostAlloc()

4.40.1.3 #define CU_MEMHOSTALLOC_WRITECOMBINED

If set, host memory is allocated as write-combined - fast to write, faster to DMA, slow to read except via SSE4 streaming load instruction (MOVNTDQA). Flag for cuMemHostAlloc()

4.40.1.4 #define CU_PARAM_TR_DEFAULT

For texture references loaded into the module, use default texunit from texture reference.

4.40.1.5 #define CU_TRSA_OVERRIDE_FORMAT

Override the texref format with a format inferred from the array. Flag for cuTexRefSetArray()

4.40.1.6 #define CU_TRSF_NORMALIZED_COORDINATES

Use normalized texture coordinates in the range [0,1) instead of [0,dim). Flag for cuTexRefSetFlags()

4.40.1.7 #define CU TRSF READ AS INTEGER

Read the texture as integers rather than promoting the values to floats in the range [0,1]. Flag for cuTexRefSetFlags()

4.40.1.8 #define CUDA_VERSION

CUDA API version number

4.40.2 Typedef Documentation

4.40.2.1 typedef enum CUaddress_mode_enum CUaddress_mode

Texture reference addressing modes

4.40.2.2 typedef enum CUarray_cubemap_face_enum CUarray_cubemap_face

Array indices for cube faces

4.40.2.3 typedef enum CUarray_format_enum CUarray_format

Array formats

4.40.2.4 typedef enum CUcomputemode_enum CUcomputemode

Compute Modes

4.40.2.5 typedef enum CUctx_flags_enum CUctx_flags

Context creation flags

4.40.2.6 typedef struct CUDA_MEMCPY2D_st CUDA_MEMCPY2D

2D memory copy parameters

4.40.2.7 typedef struct CUDA_MEMCPY3D_st CUDA_MEMCPY3D

3D memory copy parameters

4.40.2.8 typedef enum CUdevice_attribute_enum CUdevice_attribute

Device properties

4.40.2.9 typedef struct CUdevprop_st CUdevprop

Legacy device properties

4.40.2.10 typedef enum CUevent_flags_enum CUevent_flags

Event creation flags

4.40.2.11 typedef enum CUfilter_mode_enum CUfilter_mode

Texture reference filtering modes

4.40.2.12 typedef enum CUfunc_cache_enum CUfunc_cache

Function cache configurations

4.40.2.13 typedef enum CUfunction_attribute_enum CUfunction_attribute

Function properties

4.40.2.14 typedef enum CUgraphicsMapResourceFlags_enum CUgraphicsMapResourceFlags

Flags for mapping and unmapping interop resources

244 Module Documentation

4.40.2.15 typedef enum CUgraphicsRegisterFlags_enum CUgraphicsRegisterFlags

Flags to register a graphics resource

4.40.2.16 typedef enum CUjit_fallback_enum CUjit_fallback

Cubin matching fallback strategies

4.40.2.17 typedef enum CUjit_option_enum CUjit_option

Online compiler options

4.40.2.18 typedef enum CUjit_target_enum CUjit_target

Online compilation targets

4.40.2.19 typedef enum CUmemorytype_enum CUmemorytype

Memory types

4.40.2.20 typedef enum cudaError_enum CUresult

Error codes

4.40.3 Enumeration Type Documentation

4.40.3.1 enum CUaddress_mode_enum

Texture reference addressing modes

Enumerator:

```
CU_TR_ADDRESS_MODE_WRAP Wrapping address mode.
```

CU_TR_ADDRESS_MODE_CLAMP Clamp to edge address mode.

CU_TR_ADDRESS_MODE_MIRROR Mirror address mode.

4.40.3.2 enum CUarray_cubemap_face_enum

Array indices for cube faces

Enumerator:

```
CU_CUBEMAP_FACE_POSITIVE_X Positive X face of cubemap.
```

CU_CUBEMAP_FACE_NEGATIVE_X Negative X face of cubemap.

CU_CUBEMAP_FACE_POSITIVE_Y Positive Y face of cubemap.

CU_CUBEMAP_FACE_NEGATIVE_Y Negative Y face of cubemap.

CU_CUBEMAP_FACE_POSITIVE_Z Positive Z face of cubemap.

CU_CUBEMAP_FACE_NEGATIVE_Z Negative Z face of cubemap.

4.40.3.3 enum CUarray_format_enum

Array formats

Enumerator:

```
CU_AD_FORMAT_UNSIGNED_INT8 Unsigned 8-bit integers.
```

CU_AD_FORMAT_UNSIGNED_INT16 Unsigned 16-bit integers.

CU_AD_FORMAT_UNSIGNED_INT32 Unsigned 32-bit integers.

CU_AD_FORMAT_SIGNED_INT8 Signed 8-bit integers.

CU_AD_FORMAT_SIGNED_INT16 Signed 16-bit integers.

CU_AD_FORMAT_SIGNED_INT32 Signed 32-bit integers.

CU_AD_FORMAT_HALF 16-bit floating point

CU_AD_FORMAT_FLOAT 32-bit floating point

4.40.3.4 enum CUcomputemode_enum

Compute Modes

Enumerator:

- CU_COMPUTEMODE_DEFAULT Default compute mode (Multiple contexts allowed per device).
- CU_COMPUTEMODE_EXCLUSIVE Compute-exclusive mode (Only one context can be present on this device at a time).
- **CU_COMPUTEMODE_PROHIBITED** Compute-prohibited mode (No contexts can be created on this device at this time).

4.40.3.5 enum CUctx_flags_enum

Context creation flags

Enumerator:

CU_CTX_SCHED_AUTO Automatic scheduling.

CU CTX SCHED SPIN Set spin as default scheduling.

CU CTX SCHED YIELD Set yield as default scheduling.

CU_CTX_BLOCKING_SYNC Use blocking synchronization.

CU_CTX_MAP_HOST Support mapped pinned allocations.

CU_CTX_LMEM_RESIZE_TO_MAX Keep local memory allocation after launch.

4.40.3.6 enum cudaError_enum

Error codes

Enumerator:

CUDA SUCCESS No errors.

CUDA_ERROR_INVALID_VALUE Invalid value.

246 Module Documentation

- CUDA_ERROR_OUT_OF_MEMORY Out of memory.
- CUDA_ERROR_NOT_INITIALIZED Driver not initialized.
- CUDA ERROR DEINITIALIZED Driver deinitialized.
- CUDA_ERROR_NO_DEVICE No CUDA-capable device available.
- CUDA ERROR INVALID DEVICE Invalid device.
- CUDA_ERROR_INVALID_IMAGE Invalid kernel image.
- CUDA_ERROR_INVALID_CONTEXT Invalid context.
- CUDA_ERROR_CONTEXT_ALREADY_CURRENT Context already current.
- CUDA_ERROR_MAP_FAILED Map failed.
- CUDA_ERROR_UNMAP_FAILED Unmap failed.
- CUDA_ERROR_ARRAY_IS_MAPPED Array is mapped.
- CUDA_ERROR_ALREADY_MAPPED Already mapped.
- CUDA ERROR NO BINARY FOR GPU No binary for GPU.
- CUDA ERROR ALREADY ACQUIRED Already acquired.
- CUDA_ERROR_NOT_MAPPED Not mapped.
- CUDA_ERROR_NOT_MAPPED_AS_ARRAY Mapped resource not available for access as an array.
- CUDA_ERROR_NOT_MAPPED_AS_POINTER Mapped resource not available for access as a pointer.
- CUDA_ERROR_INVALID_SOURCE Invalid source.
- CUDA_ERROR_FILE_NOT_FOUND File not found.
- CUDA_ERROR_INVALID_HANDLE Invalid handle.
- CUDA_ERROR_NOT_FOUND Not found.
- CUDA_ERROR_NOT_READY CUDA not ready.
- CUDA_ERROR_LAUNCH_FAILED Launch failed.
- CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES Launch exceeded resources.
- CUDA_ERROR_LAUNCH_TIMEOUT Launch exceeded timeout.
- CUDA ERROR LAUNCH INCOMPATIBLE TEXTURING Launch with incompatible texturing.
- CUDA_ERROR_POINTER_IS_64BIT Attempted to retrieve 64-bit pointer via 32-bit API function.
- CUDA ERROR SIZE IS 64BIT Attempted to retrieve 64-bit size via 32-bit API function.
- CUDA_ERROR_UNKNOWN Unknown error.

4.40.3.7 enum CUdevice attribute enum

Device properties

Enumerator:

- CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK Maximum number of threads per block.
- CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X Maximum block dimension X.
- CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y Maximum block dimension Y.
- CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z Maximum block dimension Z.
- CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X Maximum grid dimension X.
- CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y Maximum grid dimension Y.
- CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z Maximum grid dimension Z.

- CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK Maximum shared memory available per block in bytes.
- *CU_DEVICE_ATTRIBUTE_SHARED_MEMORY_PER_BLOCK* Deprecated, use CU_DEVICE_-ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK.
- CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY Memory available on device for __constant_variables in a CUDA C kernel in bytes.
- CU_DEVICE_ATTRIBUTE_WARP_SIZE Warp size in threads.
- CU_DEVICE_ATTRIBUTE_MAX_PITCH Maximum pitch in bytes allowed by memory copies.
- CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK Maximum number of 32-bit registers available per block.
- CU_DEVICE_ATTRIBUTE_REGISTERS_PER_BLOCK Deprecated, use CU_DEVICE_ATTRIBUTE_-MAX_REGISTERS_PER_BLOCK.
- CU_DEVICE_ATTRIBUTE_CLOCK_RATE Peak clock frequency in kilohertz.
- CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT Alignment requirement for textures.
- CU_DEVICE_ATTRIBUTE_GPU_OVERLAP Device can possibly copy memory and execute a kernel concurrently.
- CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT Number of multiprocessors on device.
- CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT Specifies whether there is a run time limit on kernels.
- CU_DEVICE_ATTRIBUTE_INTEGRATED Device is integrated with host memory.
- CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY Device can map host memory into CUDA address space.
- CU_DEVICE_ATTRIBUTE_COMPUTE_MODE Compute mode (See CUcomputemode for details).
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_WIDTH Maximum 1D texture width.
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_WIDTH Maximum 2D texture width.
- CU DEVICE ATTRIBUTE MAXIMUM TEXTURE2D HEIGHT Maximum 2D texture height.
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH Maximum 3D texture width.
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT Maximum 3D texture height.
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH Maximum 3D texture depth.
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_WIDTH Maximum texture array width.
- CU DEVICE ATTRIBUTE MAXIMUM TEXTURE2D ARRAY HEIGHT Maximum texture array height.
- CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_NUMSLICES Maximum slices in a texture array.
- CU_DEVICE_ATTRIBUTE_SURFACE_ALIGNMENT Alignment requirement for surfaces.
- CU_DEVICE_ATTRIBUTE_CONCURRENT_KERNELS Device can possibly execute multiple kernels concurrently.
- CU_DEVICE_ATTRIBUTE_ECC_ENABLED Device has ECC support enabled.

4.40.3.8 enum CUevent_flags_enum

Event creation flags

Enumerator:

- CU EVENT DEFAULT Default event flag.
- CU_EVENT_BLOCKING_SYNC Event uses blocking synchronization.

248 Module Documentation

4.40.3.9 enum CUfilter mode enum

Texture reference filtering modes

Enumerator:

CU_TR_FILTER_MODE_POINT Point filter mode.CU_TR_FILTER_MODE_LINEAR Linear filter mode.

4.40.3.10 enum CUfunc cache enum

Function cache configurations

4.40.3.11 enum CUfunction_attribute_enum

Function properties

Enumerator:

- CU_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK The number of threads beyond which a launch of the function would fail. This number depends on both the function and the device on which the function is currently loaded.
- CU_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES The size in bytes of statically-allocated shared memory required by this function. This does not include dynamically-allocated shared memory requested by the user at runtime.
- CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES The size in bytes of user-allocated constant memory required by this function.
- CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES The size in bytes of thread local memory used by this function.
- CU_FUNC_ATTRIBUTE_NUM_REGS The number of registers used by each thread of this function.
- CU_FUNC_ATTRIBUTE_PTX_VERSION The PTX virtual architecture version for which the function was compiled.
- CU_FUNC_ATTRIBUTE_BINARY_VERSION The binary version for which the function was compiled.

4.40.3.12 enum CUgraphicsMapResourceFlags_enum

Flags for mapping and unmapping interop resources

4.40.3.13 enum CUgraphicsRegisterFlags_enum

Flags to register a graphics resource

4.40.3.14 enum CUjit_fallback_enum

Cubin matching fallback strategies

Enumerator:

CU_PREFER_PTX Prefer to compile ptx

CU_PREFER_BINARY Prefer to fall back to compatible binary code

4.40.3.15 enum CUjit_option_enum

Online compiler options

Enumerator:

CU_JIT_MAX_REGISTERS Max number of registers that a thread may use.

Option type: unsigned int

CU_JIT_THREADS_PER_BLOCK IN: Specifies minimum number of threads per block to target compilation for

OUT: Returns the number of threads the compiler actually targeted. This restricts the resource utilization fo the compiler (e.g. max registers) such that a block with the given number of threads should be able to launch based on register limitations. Note, this option does not currently take into account any other resource limitations, such as shared memory utilization.

Option type: unsigned int

CU_JIT_WALL_TIME Returns a float value in the option of the wall clock time, in milliseconds, spent creating the cubin

Option type: float

CU_JIT_INFO_LOG_BUFFER Pointer to a buffer in which to print any log messsages from PTXAS that are informational in nature (the buffer size is specified via option CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES)

Option type: char*

CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES IN: Log buffer size in bytes. Log messages will be capped at this size (including null terminator)

OUT: Amount of log buffer filled with messages

Option type: unsigned int

CU_JIT_ERROR_LOG_BUFFER Pointer to a buffer in which to print any log messages from PTXAS that reflect errors (the buffer size is specified via option CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES)

Option type: char*

CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES IN: Log buffer size in bytes. Log messages will be capped at this size (including null terminator)

OUT: Amount of log buffer filled with messages

Option type: unsigned int

CU_JIT_OPTIMIZATION_LEVEL Level of optimizations to apply to generated code (0 - 4), with 4 being the default and highest level of optimizations.

Option type: unsigned int

CU_JIT_TARGET_FROM_CUCONTEXT No option value required. Determines the target based on the current attached context (default)

Option type: No option value needed

CU_JIT_TARGET Target is chosen based on supplied CUjit_target_enum.

Option type: unsigned int for enumerated type CUjit_target_enum

CU_JIT_FALLBACK_STRATEGY Specifies choice of fallback strategy if matching cubin is not found. Choice is based on supplied CUjit_fallback_enum.

Option type: unsigned int for enumerated type CUjit_fallback_enum

250 Module Documentation

4.40.3.16 enum CUjit_target_enum

Online compilation targets

Enumerator:

```
CU_TARGET_COMPUTE_10 Compute device class 1.0.
CU_TARGET_COMPUTE_11 Compute device class 1.1.
CU_TARGET_COMPUTE_12 Compute device class 1.2.
CU_TARGET_COMPUTE_13 Compute device class 1.3.
CU_TARGET_COMPUTE_20 Compute device class 2.0.
```

4.40.3.17 enum CUmemorytype_enum

Memory types

Enumerator:

```
CU_MEMORYTYPE_HOST Host memory.CU_MEMORYTYPE_DEVICE Device memory.CU_MEMORYTYPE_ARRAY Array memory.
```

Chapter 5

Data Structure Documentation

5.1 CUDA_ARRAY3D_DESCRIPTOR Struct Reference

Data Fields

- unsigned int Depth

 Depth of 3D array.
- unsigned int Flags *Flags*.
- CUarray_format Format Array format.
- unsigned int Height

 Height of 3D array.
- unsigned int NumChannels

 Channels per array element.
- unsigned int Width Width of 3D array.

5.1.1 Detailed Description

3D array descriptor

5.2 CUDA_ARRAY_DESCRIPTOR Struct Reference

Data Fields

• CUarray_format Format

Array format.

• unsigned int Height

 $Height\ of\ array.$

• unsigned int NumChannels

Channels per array element.

• unsigned int Width

Width of array.

5.2.1 Detailed Description

Array descriptor

5.3 CUDA_MEMCPY2D_st Struct Reference

Data Fields

• CUarray dstArray

Destination array reference.

• CUdeviceptr dstDevice

Destination device pointer.

void * dstHost

Destination host pointer.

• CUmemorytype dstMemoryType

Destination memory type (host, device, array).

• unsigned int dstPitch

Destination pitch (ignored when dst is array).

• unsigned int dstXInBytes

Destination X in bytes.

• unsigned int dstY

Destination Y.

· unsigned int Height

Height of 2D memory copy.

CUarray srcArray

Source array reference.

• CUdeviceptr srcDevice

Source device pointer.

const void * srcHost

Source host pointer.

• CUmemorytype srcMemoryType

Source memory type (host, device, array).

• unsigned int srcPitch

Source pitch (ignored when src is array).

• unsigned int srcXInBytes

Source X in bytes.

• unsigned int srcY

Source Y.

• unsigned int WidthInBytes

Width of 2D memory copy in bytes.

5.3.1 Detailed Description

2D memory copy parameters

5.4 CUDA_MEMCPY3D_st Struct Reference

Data Fields

· unsigned int Depth

Depth of 3D memory copy.

• CUarray dstArray

Destination array reference.

• CUdeviceptr dstDevice

Destination device pointer.

• unsigned int dstHeight

Destination height (ignored when dst is array; may be 0 if Depth==1).

void * dstHost

Destination host pointer.

• unsigned int dstLOD

Destination LOD.

• CUmemorytype dstMemoryType

Destination memory type (host, device, array).

• unsigned int dstPitch

Destination pitch (ignored when dst is array).

• unsigned int dstXInBytes

Destination X in bytes.

unsigned int dstY

Destination Y.

• unsigned int dstZ

Destination Z.

• unsigned int Height

Height of 3D memory copy.

• void * reserved0

Must be NULL.

void * reserved1

Must be NULL.

CUarray srcArray

Source array reference.

• CUdeviceptr srcDevice

Source device pointer.

• unsigned int srcHeight

Source height (ignored when src is array; may be 0 if Depth==1).

• const void * srcHost

Source host pointer.

• unsigned int srcLOD

Source LOD.

• CUmemorytype srcMemoryType

Source memory type (host, device, array).

• unsigned int srcPitch

Source pitch (ignored when src is array).

• unsigned int srcXInBytes

Source X in bytes.

• unsigned int srcY

Source Y.

• unsigned int srcZ

Source Z.

• unsigned int WidthInBytes

Width of 3D memory copy in bytes.

5.4.1 Detailed Description

3D memory copy parameters

5.5 cudaChannelFormatDesc Struct Reference

Data Fields

- enum cudaChannelFormatKind f Channel format kind.
- int \mathbf{w}

w

• int x

х

• int y

y

• int z

z

5.5.1 Detailed Description

CUDA Channel format descriptor

5.6 cudaDeviceProp Struct Reference

Data Fields

• int canMapHostMemory

Device can map host memory with cudaHostAlloc/cudaHostGetDevicePointer.

• int clockRate

Clock frequency in kilohertz.

• int computeMode

Compute mode (See cudaComputeMode).

• int concurrentKernels

Device can possibly execute multiple kernels concurrently.

• int deviceOverlap

Device can concurrently copy memory and execute a kernel.

• int integrated

Device is integrated as opposed to discrete.

• int kernelExecTimeoutEnabled

Specified whether there is a run time limit on kernels.

• int major

Major compute capability.

• int maxGridSize [3]

Maximum size of each dimension of a grid.

• int maxTexture1D

Maximum 1D texture size.

• int maxTexture2D [2]

Maximum 2D texture dimensions.

• int maxTexture2DArray [3]

Maximum 2D texture array dimensions.

• int maxTexture3D [3]

Maximum 3D texture dimensions.

• int maxThreadsDim [3]

Maximum size of each dimension of a block.

int maxThreadsPerBlock

Maximum number of threads per block.

size_t memPitch

Maximum pitch in bytes allowed by memory copies.

• int minor

Minor compute capability.

• int multiProcessorCount

Number of multiprocessors on device.

• char name [256]

ASCII string identifying device.

• int regsPerBlock

32-bit registers available per block

• size_t sharedMemPerBlock

Shared memory available per block in bytes.

• size_t textureAlignment

Alignment requirement for textures.

• size_t totalConstMem

Constant memory available on device in bytes.

• size_t totalGlobalMem

Global memory available on device in bytes.

• int warpSize

Warp size in threads.

5.6.1 Detailed Description

CUDA device properties

5.7 cudaExtent Struct Reference

Data Fields

- size_t depth

 Depth in bytes.
- size_t height

 Height in bytes.
- size_t width

 Width in bytes.

5.7.1 Detailed Description

CUDA extent

See also:

make_cudaExtent

5.8 cudaFuncAttributes Struct Reference

Data Fields

• int binaryVersion

Binary architecture version for which the function was compiled. This value is the major binary version *10 + the minor binary version, so a binary version 1.3 function would return the value 13. For device emulation kernels, this is set to 9999.

• size_t constSizeBytes

Size of constant memory in bytes.

• size_t localSizeBytes

Size of local memory in bytes.

• int maxThreadsPerBlock

Maximum number of threads per block.

• int numRegs

Number of registers used.

• int ptxVersion

PTX virtual architecture version for which the function was compiled. This value is the major PTX version *10 + the minor PTX version, so a PTX version 1.3 function would return the value 13. For device emulation kernels, this is set to 9999.

• size_t sharedSizeBytes

Size of shared memory in bytes.

5.8.1 Detailed Description

CUDA function attributes

5.9 cudaMemcpy3DParms Struct Reference

Data Fields

- struct cudaArray * dstArray

 Destination memory address.
- struct cudaPos dstPos

 Destination position offset.
- struct cudaPitchedPtr dstPtr

 Pitched destination memory address.
- struct cudaExtent extent

 Requested memory copy size.
- enum cudaMemcpyKind kind Type of transfer.
- struct cudaArray * srcArray Source memory address.
- struct cudaPos srcPos

 Source position offset.
- struct cudaPitchedPtr srcPtr

 Pitched source memory address.

5.9.1 Detailed Description

CUDA 3D memory copying parameters

5.10 cudaPitchedPtr Struct Reference

Data Fields

size_t pitch

 ${\it Pitch~of~allocated~memory~in~bytes.}$

• void * ptr

Pointer to allocated memory.

• size_t xsize

Logical width of allocation in elements.

• size_t ysize

Logical height of allocation in elements.

5.10.1 Detailed Description

CUDA Pitched memory pointer

See also:

make_cudaPitchedPtr

5.11 cudaPos Struct Reference

Data Fields

- size_t x
 - \boldsymbol{x}
- size_t y
 - y
- size_t z
 - z

5.11.1 Detailed Description

CUDA 3D position

See also:

make_cudaPos

5.12 CUdevprop_st Struct Reference

Data Fields

• int clockRate

Clock frequency in kilohertz.

• int maxGridSize [3]

Maximum size of each dimension of a grid.

• int maxThreadsDim [3]

Maximum size of each dimension of a block.

• int maxThreadsPerBlock

Maximum number of threads per block.

• int memPitch

Maximum pitch in bytes allowed by memory copies.

• int regsPerBlock

32-bit registers available per block

• int sharedMemPerBlock

Shared memory available per block in bytes.

• int SIMDWidth

Warp size in threads.

• int textureAlign

Alignment requirement for textures.

• int totalConstantMemory

Constant memory available on device in bytes.

5.12.1 Detailed Description

Legacy device properties

Index

C++ API Routines, 103	CUDA_TYPES, 244
Context Management, 127	CU_CUBEMAP_FACE_POSITIVE_Z
CU_AD_FORMAT_FLOAT	CUDA_TYPES, 244
CUDA_TYPES, 245	CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST
CU_AD_FORMAT_HALF	MEMORY
CUDA_TYPES, 245	CUDA_TYPES, 247
CU_AD_FORMAT_SIGNED_INT16	CU_DEVICE_ATTRIBUTE_CLOCK_RATE
CUDA_TYPES, 245	CUDA_TYPES, 247
CU_AD_FORMAT_SIGNED_INT32	CU_DEVICE_ATTRIBUTE_COMPUTE_MODE
CUDA_TYPES, 245	CUDA_TYPES, 247
CU_AD_FORMAT_SIGNED_INT8	CU_DEVICE_ATTRIBUTE_CONCURRENT
CUDA_TYPES, 245	KERNELS
CU_AD_FORMAT_UNSIGNED_INT16	CUDA_TYPES, 247
CUDA_TYPES, 245	CU_DEVICE_ATTRIBUTE_ECC_ENABLED
CU_AD_FORMAT_UNSIGNED_INT32	CUDA_TYPES, 247
CUDA_TYPES, 245	CU_DEVICE_ATTRIBUTE_GPU_OVERLAP
CU_AD_FORMAT_UNSIGNED_INT8	CUDA_TYPES, 247
CUDA_TYPES, 245	CU_DEVICE_ATTRIBUTE_INTEGRATED
CU_COMPUTEMODE_DEFAULT	CUDA_TYPES, 247
CUDA_TYPES, 245	CU_DEVICE_ATTRIBUTE_KERNEL_EXEC
CU_COMPUTEMODE_EXCLUSIVE	TIMEOUT
CUDA_TYPES, 245	CUDA_TYPES, 247
CU_COMPUTEMODE_PROHIBITED	CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X
CUDA_TYPES, 245	CUDA_TYPES, 246
CU_CTX_BLOCKING_SYNC	CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y
CUDA_TYPES, 245	CUDA_TYPES, 246
CU_CTX_LMEM_RESIZE_TO_MAX	CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z
CUDA_TYPES, 245	CUDA_TYPES, 246
CU_CTX_MAP_HOST	CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X
CUDA_TYPES, 245	CUDA_TYPES, 246
CU_CTX_SCHED_AUTO	CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y
CUDA_TYPES, 245	CUDA_TYPES, 246
CU_CTX_SCHED_SPIN	CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z
CUDA_TYPES, 245	CUDA_TYPES, 246
CU_CTX_SCHED_YIELD	CU_DEVICE_ATTRIBUTE_MAX_PITCH
CUDA_TYPES, 245	CUDA_TYPES, 247
CU_CUBEMAP_FACE_NEGATIVE_X	CU_DEVICE_ATTRIBUTE_MAX_REGISTERS
CUDA_TYPES, 244	PER_BLOCK
CU_CUBEMAP_FACE_NEGATIVE_Y	CUDA_TYPES, 247
CUDA_TYPES, 244	CU_DEVICE_ATTRIBUTE_MAX_SHARED
CU_CUBEMAP_FACE_NEGATIVE_Z	MEMORY_PER_BLOCK
CUDA_TYPES, 244	CUDA_TYPES, 246
CU_CUBEMAP_FACE_POSITIVE_X	CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_
CUDA_TYPES, 244	BLOCK
CU_CUBEMAP_FACE_POSITIVE_Y	CUDA_TYPES, 246

CU_DEVICE_ATTRIBUTE_MAXIMUM	CUDA_TYPES, 248
TEXTURE1D_WIDTH	CU_FUNC_ATTRIBUTE_MAX_THREADS_PER
CUDA_TYPES, 247	BLOCK
CU_DEVICE_ATTRIBUTE_MAXIMUM	CUDA_TYPES, 248
TEXTURE2D_ARRAY_HEIGHT	CU_FUNC_ATTRIBUTE_NUM_REGS
CUDA_TYPES, 247	CUDA_TYPES, 248
CU_DEVICE_ATTRIBUTE_MAXIMUM	CU_FUNC_ATTRIBUTE_PTX_VERSION
TEXTURE2D_ARRAY_NUMSLICES	CUDA_TYPES, 248
CUDA_TYPES, 247	CU_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES
CU_DEVICE_ATTRIBUTE_MAXIMUM	CUDA_TYPES, 248
TEXTURE2D_ARRAY_WIDTH	CU_JIT_ERROR_LOG_BUFFER
CUDA TYPES, 247	CUDA_TYPES, 249
CU_DEVICE_ATTRIBUTE_MAXIMUM	CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES
TEXTURE2D_HEIGHT	CUDA_TYPES, 249
CUDA_TYPES, 247	CU_JIT_FALLBACK_STRATEGY
CU_DEVICE_ATTRIBUTE_MAXIMUM	CUDA_TYPES, 249
TEXTURE2D_WIDTH	CU_JIT_INFO_LOG_BUFFER
CUDA_TYPES, 247	CUDA_TYPES, 249
CU_DEVICE_ATTRIBUTE_MAXIMUM	CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES
TEXTURE3D_DEPTH	CUDA_TYPES, 249
CUDA_TYPES, 247	CU_JIT_MAX_REGISTERS
CU_DEVICE_ATTRIBUTE_MAXIMUM	CUDA_TYPES, 249
TEXTURE3D_HEIGHT	CU_JIT_OPTIMIZATION_LEVEL
CUDA_TYPES, 247	CUDA_TYPES, 249
CU_DEVICE_ATTRIBUTE_MAXIMUM	CU_JIT_TARGET
TEXTURE3D_WIDTH	CUDA_TYPES, 249
CUDA_TYPES, 247	CU_JIT_TARGET_FROM_CUCONTEXT
CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR	CUDA_TYPES, 249
COUNT	CU_JIT_THREADS_PER_BLOCK
CUDA_TYPES, 247	CUDA_TYPES, 249
CU_DEVICE_ATTRIBUTE_REGISTERS_PER	CU_JIT_WALL_TIME
BLOCK	CUDA_TYPES, 249
CUDA TYPES, 247	CU_MEMORYTYPE_ARRAY
CU_DEVICE_ATTRIBUTE_SHARED_MEMORY	CUDA_TYPES, 250
PER BLOCK	CU MEMORYTYPE DEVICE
CUDA_TYPES, 247	CUDA_TYPES, 250
CU_DEVICE_ATTRIBUTE_SURFACE_ALIGNMENT	
CUDA_TYPES, 247	CUDA_TYPES, 250
CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT	CU_PREFER_BINARY
CUDA_TYPES, 247	CUDA_TYPES, 248
CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT	CU PREFER PTX
MEMORY	-
	CUDA_TYPES, 248
CUDA_TYPES, 247	CU_TARGET_COMPUTE_10
CU_DEVICE_ATTRIBUTE_WARP_SIZE	CUDA_TYPES, 250
CUDA_TYPES, 247	CU_TARGET_COMPUTE_11
CU_EVENT_BLOCKING_SYNC	CUDA_TYPES, 250
CUDA_TYPES, 247	CU_TARGET_COMPUTE_12
CU_EVENT_DEFAULT	CUDA_TYPES, 250
CUDA_TYPES, 247	CU_TARGET_COMPUTE_13
CU_FUNC_ATTRIBUTE_BINARY_VERSION	CUDA_TYPES, 250
CUDA_TYPES, 248	CU_TARGET_COMPUTE_20
CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES	CUDA_TYPES, 250
CUDA_TYPES, 248	CU_TR_ADDRESS_MODE_CLAMP
CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES	CUDA_TYPES, 244

CU_TR_ADDRESS_MODE_MIRROR	cuCtxPopCurrent, 130
CUDA_TYPES, 244	cuCtxPushCurrent, 131
CU_TR_ADDRESS_MODE_WRAP	cuCtxSynchronize, 131
CUDA_TYPES, 244	CUctx_flags
CU_TR_FILTER_MODE_LINEAR	CUDA_TYPES, 243
CUDA_TYPES, 248	CUctx_flags_enum
CU_TR_FILTER_MODE_POINT	CUDA_TYPES, 245
CUDA_TYPES, 248	cuCtxAttach
CU_MEMHOSTALLOC_DEVICEMAP	CUCTX, 127
CUDA_TYPES, 242	cuCtxCreate
CU_MEMHOSTALLOC_PORTABLE	CUCTX, 128
CUDA_TYPES, 242	cuCtxDestroy
CU_MEMHOSTALLOC_WRITECOMBINED	CUCTX, 129
CUDA_TYPES, 242	cuCtxDetach
CU_PARAM_TR_DEFAULT	CUCTX, 129
CUDA_TYPES, 242	cuCtxGetDevice
CU_TRSA_OVERRIDE_FORMAT	CUCTX, 130
CUDA_TYPES, 242	cuCtxPopCurrent
CU_TRSF_NORMALIZED_COORDINATES	CUCTX, 130
CUDA_TYPES, 242	cuCtxPushCurrent
CU_TRSF_READ_AS_INTEGER	CUCTX, 131
CUDA_TYPES, 242	cuCtxSynchronize
CUaddress_mode	CUCTX, 131
CUDA_TYPES, 242	CUD3D10
CUaddress_mode_enum	cuD3D10CtxCreate, 217
CUDA_TYPES, 244	cuD3D10GetDevice, 218
cuArray3DCreate	cuGraphicsD3D10RegisterResource, 218
CUMEM, 154	CUD3D10_DEPRECATED
cuArray3DGetDescriptor	cuD3D10MapResources, 220
CUMEM, 156	cuD3D10RegisterResource, 221
CUarray_cubemap_face	cuD3D10ResourceGetMappedArray, 222
CUDA_TYPES, 242	cuD3D10ResourceGetMappedPitch, 223
CUarray_cubemap_face_enum	cuD3D10ResourceGetMappedPointer, 224
CUDA_TYPES, 244	cuD3D10ResourceGetMappedSize, 224
CUPA TYPES 242	cuD3D10ResourceGetSurfaceDimensions, 225
CUDA_TYPES, 242 CUarray_format_enum	cuD3D10ResourceSetMapFlags, 226
CUDA_TYPES, 244	cuD3D10UnmapResources, 227 cuD3D10UnregisterResource, 227
cuArrayCreate	cuD3D10UllregisterResource, 227 cuD3D10CtxCreate
CUMEM, 156	CUD3D10, 217
cuArrayDestroy	cuD3D10GetDevice
CUMEM, 158	CUD3D10, 218
cuArrayGetDescriptor	cuD3D10, 218 cuD3D10MapResources
CUMEM, 158	CUD3D10_DEPRECATED, 220
CUcomputemode	cuD3D10_DEFRECATED, 220
CUDA_TYPES, 243	CUD3D10_DEPRECATED, 221
CUcomputemode_enum	cuD3D10_bEl RECATED, 221 cuD3D10ResourceGetMappedArray
CUDA_TYPES, 245	CUD3D10_DEPRECATED, 222
CUCTX	cuD3D10ResourceGetMappedPitch
cuCtxAttach, 127	CUD3D10_DEPRECATED, 223
cuCtxAttach, 127 cuCtxCreate, 128	cuD3D10_bEl RECATED, 223 cuD3D10ResourceGetMappedPointer
cuCtxDestroy, 129	CUD3D10_DEPRECATED, 224
cuCtxDesitoy, 129	cuD3D10ResourceGetMappedSize
cuCtxGetDevice, 130	CUD3D10_DEPRECATED, 224
1.30.000.000, 200	

cuD3D10ResourceGetSurfaceDimensions	cuD3D9UnmapResources
CUD3D10_DEPRECATED, 225	CUD3D9_DEPRECATED, 215
cuD3D10ResourceSetMapFlags	cuD3D9UnregisterResource
CUD3D10_DEPRECATED, 226	CUD3D9_DEPRECATED, 216
cuD3D10UnmapResources	CUDA Driver API, 118
CUD3D10_DEPRECATED, 227	CUDA Runtime API, 9
cuD3D10UnregisterResource	CUDA_ERROR_ALREADY_ACQUIRED
CUD3D10_DEPRECATED, 227	CUDA_TYPES, 246
CUD3D11	CUDA_ERROR_ALREADY_MAPPED
cuD3D11CtxCreate, 229	CUDA TYPES, 246
cuD3D11GetDevice, 230	CUDA_ERROR_ARRAY_IS_MAPPED
cuGraphicsD3D11RegisterResource, 230	CUDA_TYPES, 246
cuD3D11CtxCreate	CUDA_ERROR_CONTEXT_ALREADY_CURRENT
CUD3D11, 229	CUDA_TYPES, 246
cuD3D11GetDevice	CUDA_ERROR_DEINITIALIZED
CUD3D11, 230	CUDA_TYPES, 246
CUD3D9	CUDA_FITES, 240 CUDA_ERROR_FILE_NOT_FOUND
cuD3D9CtxCreate, 205	CUDA_TYPES, 246
•	CUDA_FITES, 240 CUDA_ERROR_INVALID_CONTEXT
cuD3D9GetDevice, 206	
cuGraphicsD3D9RegisterResource, 206	CUDA_TYPES, 246
CUD3D9_DEPRECATED	CUDA_ERROR_INVALID_DEVICE
cuD3D9GetDirect3DDevice, 209	CUDA_TYPES, 246
cuD3D9MapResources, 209	CUDA_ERROR_INVALID_HANDLE
cuD3D9RegisterResource, 210	CUDA_TYPES, 246
cuD3D9ResourceGetMappedArray, 211	CUDA_ERROR_INVALID_IMAGE
cuD3D9ResourceGetMappedPitch, 212	CUDA_TYPES, 246
cuD3D9ResourceGetMappedPointer, 212	CUDA_ERROR_INVALID_SOURCE
cuD3D9ResourceGetMappedSize, 213	CUDA_TYPES, 246
cuD3D9ResourceGetSurfaceDimensions, 214	CUDA_ERROR_INVALID_VALUE
cuD3D9ResourceSetMapFlags, 215	CUDA_TYPES, 245
cuD3D9UnmapResources, 215	CUDA_ERROR_LAUNCH_FAILED
cuD3D9UnregisterResource, 216	CUDA_TYPES, 246
cuD3D9CtxCreate	CUDA_ERROR_LAUNCH_INCOMPATIBLE
CUD3D9, 205	TEXTURING
cuD3D9GetDevice	CUDA_TYPES, 246
CUD3D9, 206	CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES
cuD3D9GetDirect3DDevice	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 209	CUDA_ERROR_LAUNCH_TIMEOUT
cuD3D9MapResources	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 209	CUDA_ERROR_MAP_FAILED
cuD3D9RegisterResource	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 210	CUDA_ERROR_NO_BINARY_FOR_GPU
cuD3D9ResourceGetMappedArray	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 211	CUDA_ERROR_NO_DEVICE
cuD3D9ResourceGetMappedPitch	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 212	CUDA_ERROR_NOT_FOUND
cuD3D9ResourceGetMappedPointer	CUDA TYPES, 246
CUD3D9_DEPRECATED, 212	CUDA_ERROR_NOT_INITIALIZED
cuD3D9ResourceGetMappedSize	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 213	CUDA_ERROR_NOT_MAPPED
cuD3D9ResourceGetSurfaceDimensions	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 214	CUDA_ERROR_NOT_MAPPED_AS_ARRAY
cuD3D9ResourceSetMapFlags	CUDA_TYPES, 246
CUD3D9_DEPRECATED, 215	CUDA_ERROR_NOT_MAPPED_AS_POINTER

CUDA_TYPES, 246	CU_DEVICE_ATTRIBUTE_MAX_BLOCK
CUDA_ERROR_NOT_READY	DIM_X, 246
CUDA_TYPES, 246	CU_DEVICE_ATTRIBUTE_MAX_BLOCK
CUDA_ERROR_OUT_OF_MEMORY	DIM_Y, 246
CUDA_TYPES, 245	CU_DEVICE_ATTRIBUTE_MAX_BLOCK
CUDA_ERROR_POINTER_IS_64BIT	DIM_Z, 246
CUDA_TYPES, 246	CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM
CUDA_ERROR_SIZE_IS_64BIT	X, 246
CUDA_TYPES, 246	CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM
CUDA ERROR UNKNOWN	Y, 246
CUDA_TYPES, 246	CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM
CUDA ERROR UNMAP FAILED	Z, 246
CUDA_TYPES, 246	CU_DEVICE_ATTRIBUTE_MAX_PITCH, 247
CUDA_SUCCESS	CU_DEVICE_ATTRIBUTE_MAX
CUDA_TYPES, 245	REGISTERS_PER_BLOCK, 247
CUDA_TYPES	CU_DEVICE_ATTRIBUTE_MAX_SHARED
	MEMORY_PER_BLOCK, 246
CU_AD_FORMAT_FLOAT, 245	CU_DEVICE_ATTRIBUTE_MAX_THREADS
CU_AD_FORMAT_HALF, 245	PER BLOCK, 246
CU_AD_FORMAT_SIGNED_INT16, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_AD_FORMAT_SIGNED_INT32, 245	TEXTURE1D_WIDTH, 247
CU_AD_FORMAT_SIGNED_INT8, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_AD_FORMAT_UNSIGNED_INT16, 245	TEXTURE2D_ARRAY_HEIGHT, 247
CU_AD_FORMAT_UNSIGNED_INT32, 245	CU DEVICE ATTRIBUTE MAXIMUM -
CU_AD_FORMAT_UNSIGNED_INT8, 245	TEXTURE2D_ARRAY_NUMSLICES, 247
CU_COMPUTEMODE_DEFAULT, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_COMPUTEMODE_EXCLUSIVE, 245	TEXTURE2D_ARRAY_WIDTH, 247
CU_COMPUTEMODE_PROHIBITED, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_CTX_BLOCKING_SYNC, 245	TEXTURE2D_HEIGHT, 247
CU_CTX_LMEM_RESIZE_TO_MAX, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_CTX_MAP_HOST, 245	TEXTURE2D_WIDTH, 247
CU_CTX_SCHED_AUTO, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_CTX_SCHED_SPIN, 245	TEXTURE3D_DEPTH, 247
CU_CTX_SCHED_YIELD, 245	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_CUBEMAP_FACE_NEGATIVE_X, 244	TEXTURE3D_HEIGHT, 247
CU_CUBEMAP_FACE_NEGATIVE_Y, 244	CU_DEVICE_ATTRIBUTE_MAXIMUM
CU_CUBEMAP_FACE_NEGATIVE_Z, 244	TEXTURE3D_WIDTH, 247
CU_CUBEMAP_FACE_POSITIVE_X, 244	CU_DEVICE_ATTRIBUTE
CU CUBEMAP FACE POSITIVE Y, 244	MULTIPROCESSOR_COUNT, 247
CU_CUBEMAP_FACE_POSITIVE_Z, 244	CU_DEVICE_ATTRIBUTE_REGISTERS_PER
CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST	BLOCK, 247
MEMORY, 247	
CU_DEVICE_ATTRIBUTE_CLOCK_RATE, 247	CU_DEVICE_ATTRIBUTE_SHARED MEMORY PER BLOCK, 247
CU_DEVICE_ATTRIBUTE_CLOCK_KATE, 247 CU_DEVICE_ATTRIBUTE_COMPUTE_MODE,	<u> </u>
247	CU_DEVICE_ATTRIBUTE_SURFACE
	ALIGNMENT, 247
CU_DEVICE_ATTRIBUTE_CONCURRENT	CU_DEVICE_ATTRIBUTE_TEXTURE
KERNELS, 247	ALIGNMENT, 247
CU_DEVICE_ATTRIBUTE_ECC_ENABLED,	CU_DEVICE_ATTRIBUTE_TOTAL
247	CONSTANT_MEMORY, 247
CU_DEVICE_ATTRIBUTE_GPU_OVERLAP,	CU_DEVICE_ATTRIBUTE_WARP_SIZE, 247
247	CU_EVENT_BLOCKING_SYNC, 247
CU_DEVICE_ATTRIBUTE_INTEGRATED, 247	CU_EVENT_DEFAULT, 247
CU_DEVICE_ATTRIBUTE_KERNEL_EXEC	CU_FUNC_ATTRIBUTE_BINARY_VERSION,
TIMEOUT, 247	248

CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES, 248	CUDA_ERROR_LAUNCH_OUT_OF RESOURCES, 246
CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES,	CUDA_ERROR_LAUNCH_TIMEOUT, 246
248	CUDA_ERROR_MAP_FAILED, 246
CU_FUNC_ATTRIBUTE_MAX_THREADS	CUDA_ERROR_NO_BINARY_FOR_GPU, 246
PER_BLOCK, 248	CUDA_ERROR_NO_DEVICE, 246
CU_FUNC_ATTRIBUTE_NUM_REGS, 248	CUDA_ERROR_NOT_FOUND, 246
CU_FUNC_ATTRIBUTE_PTX_VERSION, 248	CUDA_ERROR_NOT_INITIALIZED, 246
CU_FUNC_ATTRIBUTE_SHARED_SIZE	CUDA_ERROR_NOT_MAPPED, 246
BYTES, 248	CUDA_ERROR_NOT_MAPPED_AS_ARRAY,
CU_JIT_ERROR_LOG_BUFFER, 249	246
CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES,	CUDA_ERROR_NOT_MAPPED_AS_POINTER,
249	246
CU_JIT_FALLBACK_STRATEGY, 249	CUDA_ERROR_NOT_READY, 246
CU_JIT_INFO_LOG_BUFFER, 249	CUDA_ERROR_OUT_OF_MEMORY, 245
CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES,	CUDA_ERROR_POINTER_IS_64BIT, 246
249	CUDA_ERROR_SIZE_IS_64BIT, 246
CU_JIT_MAX_REGISTERS, 249	CUDA_ERROR_UNKNOWN, 246
CU_JIT_OPTIMIZATION_LEVEL, 249	CUDA_ERROR_UNMAP_FAILED, 246
CU_JIT_TARGET, 249	CUDA_SUCCESS, 245
CU_JIT_TARGET_FROM_CUCONTEXT, 249	CUDA_ARRAY3D_DESCRIPTOR, 251
CU_JIT_THREADS_PER_BLOCK, 249	CUDA_ARRAY_DESCRIPTOR, 252
CU_JIT_WALL_TIME, 249	CUDA_MEMCPY2D
CU_MEMORYTYPE_ARRAY, 250	CUDA_TYPES, 243
CU_MEMORYTYPE_DEVICE, 250	CUDA_MEMCPY2D_st, 253
CU_MEMORYTYPE_HOST, 250	CUDA_MEMCPY3D
CU_PREFER_BINARY, 248	CUDA_TYPES, 243
CU_PREFER_PTX, 248	CUDA_MEMCPY3D_st, 255
CU_TARGET_COMPUTE_10, 250	CUDA_TYPES
CU_TARGET_COMPUTE_11, 250	CU_MEMHOSTALLOC_DEVICEMAP, 242
CU_TARGET_COMPUTE_12, 250	CU_MEMHOSTALLOC_PORTABLE, 242
CU_TARGET_COMPUTE_13, 250	CU_MEMHOSTALLOC_WRITECOMBINED,
CU TARGET COMPUTE 20, 250	242
CU_TR_ADDRESS_MODE_CLAMP, 244	CU_PARAM_TR_DEFAULT, 242
CU_TR_ADDRESS_MODE_MIRROR, 244	CU_TRSA_OVERRIDE_FORMAT, 242
CU_TR_ADDRESS_MODE_WRAP, 244	CU_TRSF_NORMALIZED_COORDINATES, 242
CU_TR_FILTER_MODE_LINEAR, 248	CU_TRSF_READ_AS_INTEGER, 242
CU_TR_FILTER_MODE_POINT, 248	CUaddress_mode, 242
CUDA_ERROR_ALREADY_ACQUIRED, 246	CUaddress_mode_enum, 244
CUDA_ERROR_ALREADY_MAPPED, 246	CUarray_cubemap_face, 242
CUDA_ERROR_ARRAY_IS_MAPPED, 246	CUarray_cubemap_face_enum, 244
CUDA_ERROR_CONTEXT_ALREADY	CUarray_format, 242
CURRENT, 246	CUarray_format_enum, 244
CUDA_ERROR_DEINITIALIZED, 246	CUcomputemode, 243
CUDA_ERROR_FILE_NOT_FOUND, 246	CUcomputemode_enum, 245
CUDA_ERROR_INVALID_CONTEXT, 246	CUctx_flags, 243
CUDA_ERROR_INVALID_DEVICE, 246	CUctx_flags_enum, 245
CUDA_ERROR_INVALID_HANDLE, 246	CUDA_MEMCPY2D, 243
CUDA_ERROR_INVALID_IMAGE, 246	CUDA_MEMCPY3D, 243
CUDA_ERROR_INVALID_SOURCE, 246	CUDA_VERSION, 242
CUDA_ERROR_INVALID_VALUE, 245	cudaError_enum, 245
CUDA_ERROR_LAUNCH_FAILED, 246	CUdevice_attribute, 243
CUDA_ERROR_LAUNCH_INCOMPATIBLE	CUdevice_attribute_enum, 246
TEXTURING, 246	CUdevprop, 243

CUevent_flags, 243	CUDART_EXECUTION, 25
CUevent_flags_enum, 247	cudaCreateChannelDesc
CUfilter_mode, 243	CUDART_HIGHLEVEL, 107
CUfilter_mode_enum, 247	CUDART_TEXTURE, 99
CUfunc_cache, 243	cudaD3D10GetDevice
CUfunc_cache_enum, 248	CUDART_D3D10, 79
CUfunction_attribute, 243	cudaD3D10MapResources
CUfunction_attribute_enum, 248	CUDART_D3D10_DEPRECATED, 82
CUgraphicsMapResourceFlags, 243	cudaD3D10RegisterResource
CUgraphicsMapResourceFlags_enum, 248	CUDART_D3D10_DEPRECATED, 83
CUgraphicsRegisterFlags, 243	cudaD3D10ResourceGetMappedArray
CUgraphicsRegisterFlags_enum, 248	CUDART_D3D10_DEPRECATED, 84
CUjit_fallback, 244	cudaD3D10ResourceGetMappedPitch
CUjit_fallback_enum, 248	CUDART_D3D10_DEPRECATED, 85
CUjit_option, 244	cudaD3D10ResourceGetMappedPointer
CUjit_option_enum, 248	CUDART_D3D10_DEPRECATED, 80
CUjit_target, 244	cudaD3D10ResourceGetMappedSize
CUjit_target_enum, 249	CUDART_D3D10_DEPRECATED, 80
CUmemorytype, 244	cudaD3D10ResourceGetSurfaceDimensions
7 7 2	
CUmemorytype_enum, 250	CUDART_D3D10_DEPRECATED, 87
CUresult, 244	cudaD3D10ResourceSetMapFlags
CUDA_VERSION	CUDART_D3D10_DEPRECATED, 8°
CUDA_TYPES, 242	cudaD3D10SetDirect3DDevice
cudaBindTexture	CUDART_D3D10, 79
CUDART_HIGHLEVEL, 104	cudaD3D10UnmapResources
CUDART_TEXTURE, 97	CUDART_D3D10_DEPRECATED, 88
cudaBindTexture2D	cudaD3D10UnregisterResource
CUDART_HIGHLEVEL, 105	CUDART_D3D10_DEPRECATED, 89
CUDART_TEXTURE, 98	cudaD3D11GetDevice
cudaBindTextureToArray	CUDART_D3D11, 90
CUDART_HIGHLEVEL, 106	cudaD3D11SetDirect3DDevice
CUDART_TEXTURE, 99	CUDART_D3D11, 90
cudaChannelFormatDesc, 257	cudaD3D9GetDevice
cudaChannelFormatKind	CUDART_D3D9, 67
CUDART_TYPES, 115	cudaD3D9GetDirect3DDevice
cudaChannelFormatKindFloat	CUDART_D3D9_DEPRECATED, 71
CUDART_TYPES, 115	cudaD3D9MapResources
cudaChannelFormatKindNone	CUDART_D3D9_DEPRECATED, 71
CUDART_TYPES, 115	cudaD3D9RegisterResource
cudaChannelFormatKindSigned	CUDART_D3D9_DEPRECATED, 71
CUDART_TYPES, 115	cudaD3D9ResourceGetMappedArray
cudaChannelFormatKindUnsigned	CUDART_D3D9_DEPRECATED, 73
CUDART_TYPES, 115	cudaD3D9ResourceGetMappedPitch
cudaChooseDevice	CUDART_D3D9_DEPRECATED, 73
CUDART_DEVICE, 13	cudaD3D9ResourceGetMappedPointer
cudaComputeMode	CUDART_D3D9_DEPRECATED, 74
CUDART_TYPES, 115	cudaD3D9ResourceGetMappedSize
cudaComputeModeDefault	CUDART_D3D9_DEPRECATED, 75
CUDART_TYPES, 115	cudaD3D9ResourceGetSurfaceDimensions
cudaComputeModeExclusive	CUDART_D3D9_DEPRECATED, 75
CUDART_TYPES, 115	cudaD3D9ResourceSetMapFlags
cudaComputeModeProhibited	CUDART_D3D9_DEPRECATED, 76
CUDART_TYPES, 115	cudaD3D9SetDirect3DDevice
cudaConfigureCall	CUDART_D3D9, 67
. O	 ,'

cudaD3D9UnmapResources	CUDART_TYPES, 116
CUDART_D3D9_DEPRECATED, 77	cudaErrorLaunchFailure
cudaD3D9UnregisterResource	CUDART_TYPES, 116
CUDART_D3D9_DEPRECATED, 77	cudaErrorLaunchOutOfResources
cudaDeviceProp, 258	CUDART TYPES, 116
cudaDriverGetVersion	cudaErrorLaunchTimeout
CUDART_VERSION, 102	CUDART_TYPES, 116
cudaError	cudaErrorMapBufferObjectFailed
CUDART_TYPES, 115	CUDART_TYPES, 116
cudaError_enum	cudaErrorMemoryAllocation
CUDA_TYPES, 245	CUDART_TYPES, 115
cudaError_t	cudaErrorMemoryValueTooLarge
CUDART_TYPES, 115	CUDART_TYPES, 116
cudaErrorAddressOfConstant	cudaErrorMissingConfiguration
CUDART_TYPES, 116	CUDART_TYPES, 115
	cudaErrorMixedDeviceExecution
cudaErrorApiFailureBase	
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorCudartUnloading	cudaErrorNoDevice
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorECCUncorrectable	cudaErrorNotReady
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInitializationError	cudaErrorNotYetImplemented
CUDART_TYPES, 115	CUDART_TYPES, 116
cudaErrorInsufficientDriver	cudaErrorPriorLaunchFailure
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidChannelDescriptor	cudaErrorSetOnActiveProcess
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidConfiguration	cudaErrorStartupFailure
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidDevice	cudaErrorSynchronizationError
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidDeviceFunction	cudaErrorTextureFetchFailed
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidDevicePointer	cudaErrorTextureNotBound
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidFilterSetting	cudaErrorUnknown
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidHostPointer	cudaErrorUnmapBufferObjectFailed
CUDART_TYPES, 116	CUDART_TYPES, 116
cudaErrorInvalidMemcpyDirection	cudaEvent t
CUDART_TYPES, 116	CUDART_TYPES, 115
cudaErrorInvalidNormSetting	cudaEventCreate
CUDART TYPES, 116	CUDART_EVENT, 21
cudaErrorInvalidPitchValue	cudaEventCreateWithFlags
CUDART_TYPES, 116	CUDART_EVENT, 21
cudaErrorInvalidResourceHandle	cudaEventDestroy
CUDART_TYPES, 116	CUDART_EVENT, 22
cudaErrorInvalidSymbol	cudaEventElapsedTime
CUDART_TYPES, 116	CUDART_EVENT, 22
cudaErrorInvalidTexture	cudaEventQuery
CUDART_TYPES, 116	CUDART_EVENT, 23
cudaErrorInvalidTextureBinding	cudaEventRecord
CUDART_TYPES, 116	CUDART_EVENT, 23
cudaErrorInvalidValue	cudaEventSynchronize

CUDART_EVENT, 24	cudaGLSetGLDevice
cudaExtent, 260	CUDART_OPENGL, 59
cudaFree	cudaGLUnmapBufferObject
CUDART_MEMORY, 32	CUDART_OPENGL_DEPRECATED, 64
cudaFreeArray	cudaGLUnmapBufferObjectAsync
CUDART_MEMORY, 32	CUDART_OPENGL_DEPRECATED, 65
cudaFreeHost	cudaGLUnregisterBufferObject
CUDART_MEMORY, 32	CUDART_OPENGL_DEPRECATED, 65
cudaFuncAttributes, 261	cudaGraphicsCubeFace
cudaFuncCache	CUDART_TYPES, 117
CUDART_TYPES, 116	cudaGraphicsCubeFaceNegativeX
cudaFuncCachePreferL1	CUDART_TYPES, 117
CUDART_TYPES, 117	cudaGraphicsCubeFaceNegativeY
cudaFuncCachePreferNone	CUDART_TYPES, 117
CUDART_TYPES, 117	cudaGraphicsCubeFaceNegativeZ
cudaFuncCachePreferShared	CUDART_TYPES, 117
CUDART_TYPES, 117	cudaGraphicsCubeFacePositiveX
cudaFuncGetAttributes	CUDART_TYPES, 117
CUDART_EXECUTION, 26	cudaGraphicsCubeFacePositiveY
CUDART_HIGHLEVEL, 107	CUDART_TYPES, 117
cudaFuncSetCacheConfig	cudaGraphicsCubeFacePositiveZ
CUDART_EXECUTION, 26	CUDART_TYPES, 117
CUDART_HIGHLEVEL, 108	cudaGraphicsD3D10RegisterResource
cudaGetChannelDesc	CUDART_D3D10, 80
CUDART_TEXTURE, 100	cudaGraphicsD3D11RegisterResource
cudaGetDevice	CUDART_D3D11, 91
CUDART_DEVICE, 13	cudaGraphicsD3D9RegisterResource
cudaGetDeviceCount	CUDART_D3D9, 68
CUDART_DEVICE, 14	cudaGraphicsGLRegisterBuffer
cudaGetDeviceProperties	CUDART_OPENGL, 59
CUDART_DEVICE, 14	cudaGraphicsGLRegisterImage
cudaGetErrorString	CUDART_OPENGL, 60
CUDART_ERROR, 11	cudaGraphicsMapFlags
cudaGetLastError	CUDART_TYPES, 117
CUDART_ERROR, 11	cudaGraphicsMapFlagsNone
cudaGetSymbolAddress	CUDART_TYPES, 117
CUDART_HIGHLEVEL, 108	cudaGraphicsMapFlagsReadOnly
CUDART_MEMORY, 33	CUDART_TYPES, 117
cudaGetSymbolSize	cudaGraphicsMapFlagsWriteDiscard
CUDART_HIGHLEVEL, 109	CUDART_TYPES, 117
CUDART_MEMORY, 33	cudaGraphicsMapResources
cudaGetTextureAlignmentOffset	CUDART INTEROP, 93
CUDART_HIGHLEVEL, 109	cudaGraphicsRegisterFlags
CUDART_TEXTURE, 100	CUDART_TYPES, 117
cudaGetTextureReference	cudaGraphicsRegisterFlagsNone
CUDART_TEXTURE, 101	CUDART_TYPES, 117
cudaGLMapBufferObject	cudaGraphicsResourceGetMappedPointer
CUDART OPENGL DEPRECATED, 62	CUDART_INTEROP, 94
cudaGLMapBufferObjectAsync	cudaGraphicsResourceSetMapFlags
CUDART_OPENGL_DEPRECATED, 63	CUDART_INTEROP, 94
cudaGLRegisterBufferObject	cudaGraphicsSubResourceGetMappedArray
CUDART_OPENGL_DEPRECATED, 63	CUDART_INTEROP, 95
cudaGLSetBufferObjectMapFlags	cudaGraphicsUnmapResources
CUDART OPENGL DEPRECATED, 64	CUDART INTEROP, 95

cudaGraphicsUnregisterResource	cudaMemcpyFromArrayAsync
CUDART_INTEROP, 96	CUDART_MEMORY, 50
cudaHostAlloc	cudaMemcpyFromSymbol
CUDART_MEMORY, 34	CUDART_MEMORY, 51
cudaHostGetDevicePointer	cudaMemcpyFromSymbolAsync
CUDART_MEMORY, 34	CUDART_MEMORY, 51
cudaHostGetFlags	cudaMemcpyHostToDevice
CUDART_MEMORY, 35	CUDART_TYPES, 117
cudaLaunch	cudaMemcpyHostToHost
CUDART_EXECUTION, 27	CUDART_TYPES, 117
CUDART_HIGHLEVEL, 110	cudaMemcpyKind
cudaMalloc	CUDART TYPES, 117
CUDART_MEMORY, 35	cudaMemcpyToArray
cudaMalloc3D	CUDART_MEMORY, 52
CUDART_MEMORY, 36	cudaMemcpyToArrayAsync
cudaMalloc3DArray	CUDART_MEMORY, 53
· · · · · · · · · · · · · · · · · · ·	cudaMemcpyToSymbol
CUDART_MEMORY, 36	100
cudaMallocArray	CUDART_MEMORY, 54
CUDART_MEMORY, 37	cudaMemcpyToSymbolAsync
cudaMallocHost	CUDART_MEMORY, 54
CUDART_MEMORY, 38	cudaMemGetInfo
cudaMallocPitch	CUDART_MEMORY, 55
CUDART_MEMORY, 38	cudaMemset
cudaMemcpy	CUDART_MEMORY, 55
CUDART_MEMORY, 39	cudaMemset2D
cudaMemcpy2D	CUDART_MEMORY, 56
CUDART_MEMORY, 40	cudaMemset3D
cudaMemcpy2DArrayToArray	CUDART_MEMORY, 56
CUDART_MEMORY, 40	cudaPitchedPtr, 263
cudaMemcpy2DAsync	cudaPos, 264
CUDART_MEMORY, 41	CUDART_TYPES
cudaMemcpy2DFromArray	cudaChannelFormatKindFloat, 115
CUDART_MEMORY, 42	cudaChannelFormatKindNone, 115
cudaMemcpy2DFromArrayAsync	cudaChannelFormatKindSigned, 115
CUDART_MEMORY, 43	cudaChannelFormatKindUnsigned, 115
cudaMemcpy2DToArray	cudaComputeModeDefault, 115
CUDART_MEMORY, 44	cudaComputeModeExclusive, 115
cudaMemcpy2DToArrayAsync	cudaComputeModeProhibited, 115
CUDART_MEMORY, 44	cudaErrorAddressOfConstant, 116
	cudaErrorApiFailureBase, 116
cudaMemcpy3D	<u>*</u>
CUDART_MEMORY, 45	cudaErrorCudartUnloading, 116
cudaMemcpy3DAsync	cudaErrorECCUncorrectable, 116
CUDART_MEMORY, 47	cudaErrorInitializationError, 115
cudaMemcpy3DParms, 262	cudaErrorInsufficientDriver, 116
cudaMemcpyArrayToArray	cudaErrorInvalidChannelDescriptor, 116
CUDART_MEMORY, 48	cudaErrorInvalidConfiguration, 116
cudaMemcpyAsync	cudaErrorInvalidDevice, 116
CUDART_MEMORY, 49	cudaErrorInvalidDeviceFunction, 116
cudaMemcpyDeviceToDevice	cudaErrorInvalidDevicePointer, 116
CUDART_TYPES, 117	cudaErrorInvalidFilterSetting, 116
cudaMemcpyDeviceToHost	cudaErrorInvalidHostPointer, 116
CUDART_TYPES, 117	cudaErrorInvalidMemcpyDirection, 116
cudaMemcpyFromArray	cudaErrorInvalidNormSetting, 116
CUDART_MEMORY, 49	cudaErrorInvalidPitchValue, 116
, ./	

cudaErrorInvalidResourceHandle, 116	cudaD3D10ResourceSetMapFlags, 87
cudaErrorInvalidSymbol, 116	cudaD3D10UnmapResources, 88
cudaErrorInvalidTexture, 116	cudaD3D10UnregisterResource, 89
cudaErrorInvalidTextureBinding, 116	CUDART_D3D11
cudaErrorInvalidValue, 116	cudaD3D11GetDevice, 90
cudaErrorLaunchFailure, 116	cudaD3D11SetDirect3DDevice, 90
cudaErrorLaunchOutOfResources, 116	cudaGraphicsD3D11RegisterResource, 91
cudaErrorLaunchTimeout, 116	CUDART_D3D9
cudaErrorMapBufferObjectFailed, 116	cudaD3D9GetDevice, 67
cudaErrorMemoryAllocation, 115	cudaD3D9SetDirect3DDevice, 67
cudaErrorMemoryValueTooLarge, 116	cudaGraphicsD3D9RegisterResource, 68
cudaErrorMissingConfiguration, 115	CUDART_D3D9_DEPRECATED
cudaErrorMixedDeviceExecution, 116	cudaD3D9GetDirect3DDevice, 71
cudaErrorNoDevice, 116	cudaD3D9MapResources, 71
cudaErrorNotReady, 116	cudaD3D9RegisterResource, 71
cudaErrorNotYetImplemented, 116	cudaD3D9ResourceGetMappedArray, 73
cudaErrorPriorLaunchFailure, 116	cudaD3D9ResourceGetMappedPitch, 73
cudaErrorSetOnActiveProcess, 116	cudaD3D9ResourceGetMappedPointer, 74
cudaErrorStartupFailure, 116	cudaD3D9ResourceGetMappedSize, 75
cudaErrorSynchronizationError, 116	cudaD3D9ResourceGetSurfaceDimensions, 75
cudaErrorTextureFetchFailed, 116	cudaD3D9ResourceSetMapFlags, 76
cudaErrorTextureNotBound, 116	cudaD3D9UnmapResources, 77
cudaErrorUnknown, 116	cudaD3D9UnregisterResource, 77
cudaErrorUnmapBufferObjectFailed, 116	CUDART_DEVICE
cudaFuncCachePreferL1, 117	cudaChooseDevice, 13
cudaFuncCachePreferNone, 117	cudaGetDevice, 13
cudaFuncCachePreferShared, 117	cudaGetDeviceCount, 14
cudaGraphicsCubeFaceNegativeX, 117	cudaGetDeviceProperties, 14
cudaGraphicsCubeFaceNegativeY, 117	cudaSetDevice, 16
cudaGraphicsCubeFaceNegativeZ, 117	cudaSetDeviceFlags, 16
cudaGraphicsCubeFacePositiveX, 117	cudaSetValidDevices, 17
cudaGraphicsCubeFacePositiveY, 117	CUDART_ERROR
cudaGraphicsCubeFacePositiveZ, 117	cudaGetErrorString, 11
cudaGraphicsMapFlagsNone, 117	cudaGetLastError, 11
cudaGraphicsMapFlagsReadOnly, 117	CUDART_EVENT
cudaGraphicsMapFlagsWriteDiscard, 117	cudaEventCreate, 21
cudaGraphicsRegisterFlagsNone, 117	cudaEventCreateWithFlags, 21
cudaMemcpyDeviceToDevice, 117	cudaEventDestroy, 22
cudaMemcpyDeviceToHost, 117	cudaEventElapsedTime, 22
cudaMemcpyHostToDevice, 117	cudaEventQuery, 23
cudaMemcpyHostToHost, 117	cudaEventRecord, 23
cudaSuccess, 115	cudaEventSynchronize, 24
CUDART_D3D10	CUDART_EXECUTION
cudaD3D10GetDevice, 79	cudaConfigureCall, 25
cudaD3D10SetDirect3DDevice, 79	cudaFuncGetAttributes, 26
cudaGraphicsD3D10RegisterResource, 80	cudaFuncSetCacheConfig, 26
CUDART_D3D10_DEPRECATED	cudaLaunch, 27
cudaD3D10MapResources, 82	cudaSetDoubleForDevice, 27
cudaD3D10RegisterResource, 83	cudaSetDoubleForHost, 27
cudaD3D10ResourceGetMappedArray, 84	cudaSetupArgument, 28
cudaD3D10ResourceGetMappedPitch, 85	CUDART_HIGHLEVEL
cudaD3D10ResourceGetMappedPointer, 86	cudaBindTexture, 104
cudaD3D10ResourceGetMappedSize, 86	cudaBindTexture2D, 105
cudaD3D10ResourceGetSurfaceDimensions, 87	cudaBindTextureToArray, 106

cudaCreateChannelDesc, 107	cudaMemset3D, 56
cudaFuncGetAttributes, 107	make_cudaExtent, 57
cudaFuncSetCacheConfig, 108	make_cudaPitchedPtr, 57
cudaGetSymbolAddress, 108	make_cudaPos, 57
cudaGetSymbolSize, 109	CUDART_OPENGL
cudaGetTextureAlignmentOffset, 109	cudaGLSetGLDevice, 59
cudaLaunch, 110	cudaGraphicsGLRegisterBuffer, 59
cudaSetupArgument, 110	cudaGraphicsGLRegisterImage, 60
cudaUnbindTexture, 110	cudaWGLGetDevice, 61
CUDART_INTEROP	CUDART_OPENGL_DEPRECATED
cudaGraphicsMapResources, 93	cudaGLMapBufferObject, 62
cudaGraphicsResourceGetMappedPointer, 94	cudaGLMapBufferObjectAsync, 63
cudaGraphicsResourceSetMapFlags, 94	cudaGLRegisterBufferObject, 63
cudaGraphicsSubResourceGetMappedArray, 9	
cudaGraphicsUnmapResources, 95	cudaGLUnmapBufferObject, 64
cudaGraphicsUnregisterResource, 96	cudaGLUnmapBufferObjectAsync, 65
CUDART_MEMORY	cudaGLUnregisterBufferObject, 65
cudaFree, 32	CUDART_STREAM
cudaFreeArray, 32	cudaStreamCreate, 19
cudaFreeHost, 32	cudaStreamDestroy, 19
cudaGetSymbolAddress, 33	cudaStreamQuery, 20
cudaGetSymbolAddress, 33	cudaStreamSynchronize, 20
cudaHostAlloc, 34	CUDART_TEXTURE
cudaHostGetDevicePointer, 34	cudaBindTexture, 97
cudaHostGetFlags, 35	cudaBindTexture2D, 98
cudaMalloc, 35	cudaBindTextureToArray, 99
cudaMalloc3D, 36	cudaCreateChannelDesc, 99
cudaMalloc3DArray, 36	cudaGetChannelDesc, 100
cudaMallocArray, 37	cudaGetTextureAlignmentOffset, 100
cudaMallocHost, 38	cudaGetTextureReference, 101
cudaMallocPitch, 38	cudaUnbindTexture, 101
cudaMemcpy, 39	CUDART_THREAD
cudaMemcpy2D, 40	cudaThreadExit, 10
cudaMemcpy2DArrayToArray, 40	cudaThreadSynchronize, 10
cudaMemcpy2DAsync, 41	CUDART_TYPES
cudaMemcpy2DFromArray, 42	cudaChannelFormatKind, 115
cudaMemcpy2DFromArrayAsync, 43	cudaComputeMode, 115
cudaMemcpy2DToArray, 44	cudaError, 115
cudaMemcpy2DToArrayAsync, 44	cudaError_t, 115
cudaMemcpy3D, 45	cudaEvent_t, 115
cudaMemcpy3DAsync, 47	cudaFuncCache, 116
cudaMemcpyArrayToArray, 48	cudaGraphicsCubeFace, 117
cudaMemcpyAsync, 49	cudaGraphicsMapFlags, 117
cudaMemcpyFromArray, 49	cudaGraphicsRegisterFlags, 117
cudaMemcpyFromArrayAsync, 50	cudaMemcpyKind, 117
cudaMemcpyFromSymbol, 51	cudaStream_t, 115
cudaMemcpyFromSymbolAsync, 51	CUDART_VERSION
cudaMemcpyToArray, 52	cudaDriverGetVersion, 102
cudaMemcpyToArrayAsync, 53	cudaRuntimeGetVersion, 102
cudaMemcpyToSymbol, 54	cudaRuntimeGetVersion
cudaMemcpyToSymbolAsync, 54	CUDART_VERSION, 102
cudaMemGetInfo, 55	cudaSetDevice
cudaMemset, 55	CUDART_DEVICE, 16
cudaMemset2D, 56	cudaSetDeviceFlags

CUDART_DEVICE, 16	CUDEVICE, 123
cudaSetDoubleForDevice	cuDeviceTotalMem
CUDART_EXECUTION, 27	CUDEVICE, 124
cudaSetDoubleForHost	CUdevprop
CUDART_EXECUTION, 27	CUDA_TYPES, 243
cudaSetupArgument	CUdevprop_st, 265
CUDART_EXECUTION, 28	cuDriverGetVersion
CUDART_HIGHLEVEL, 110	CUVERSION, 126
cudaSetValidDevices	CUEVENT 140
CUDART_DEVICE, 17	cuEventCreate, 140
cudaStream_t	cuEventDestroy, 141
CUDART_TYPES, 115	cuEventElapsedTime, 141
cudaStreamCreate	cuEventQuery, 141
CUDART_STREAM, 19	cuEventRecord, 142
cudaStreamDestroy CUDART_STREAM, 19	cuEventSynchronize, 142 CUevent_flags
cudaStreamQuery	CUDA_TYPES, 243
CUDART_STREAM, 20	CUevent_flags_enum
cudaStreamSynchronize	CUDA_TYPES, 247
CUDART STREAM, 20	cuEventCreate
cudaSuccess	CUEVENT, 140
CUDART_TYPES, 115	cuEventDestroy
cudaThreadExit	CUEVENT, 141
CUDART_THREAD, 10	cuEventElapsedTime
cudaThreadSynchronize	CUEVENT, 141
CUDART_THREAD, 10	cuEventQuery
cudaUnbindTexture	CUEVENT, 141
CUDART_HIGHLEVEL, 110	cuEventRecord
CUDART_TEXTURE, 101	CUEVENT, 142
cudaWGLGetDevice	cuEventSynchronize
CUDART_OPENGL, 61	CUEVENT, 142
CUDEVICE	CUEXEC
cuDeviceComputeCapability, 120	cuFuncGetAttribute, 145
cuDeviceGet, 121	cuFuncSetBlockShape, 145
cuDeviceGetAttribute, 121	cuFuncSetCacheConfig, 146
cuDeviceGetCount, 122	cuFuncSetSharedSize, 146
cuDeviceGetName, 123	cuLaunch, 147
cuDeviceGetProperties, 123	cuLaunchGrid, 147
cuDeviceTotalMem, 124	cuLaunchGridAsync, 148
CUdevice_attribute	cuParamSetf, 148
CUDA_TYPES, 243	cuParamSeti, 149
CUdevice_attribute_enum	cuParamSetSize, 149
CUDA_TYPES, 246	cuParamSetTexRef, 150
cuDeviceComputeCapability	cuParamSetv, 150
CUDEVICE, 120	CUfilter_mode
cuDeviceGet	CUDA_TYPES, 243
CUDEVICE, 121	CUfilter_mode_enum
cuDeviceGetAttribute	CUDA_TYPES, 247
CUDEVICE, 121	CUfunc_cache
cuDeviceGetCount	CUDA_TYPES, 243
CUDEVICE, 122	CUfunc_cache_enum
cuDeviceGetName	CUDA_TYPES, 248
CUDEVICE, 123	cuFuncGetAttribute
cuDeviceGetProperties	CUEXEC, 145

cuFuncSetBlockShape	CUD3D9, 206
CUEXEC, 145	cuGraphicsGLRegisterBuffer
cuFuncSetCacheConfig	CUGL, 197
CUEXEC, 146	cuGraphicsGLRegisterImage
cuFuncSetSharedSize	CUGL, 197
CUEXEC, 146	CUgraphicsMapResourceFlags
CUfunction_attribute	CUDA_TYPES, 243
CUDA_TYPES, 243	CUDA TYPES 248
CUfunction_attribute_enum	CUDA_TYPES, 248
CUDA_TYPES, 248	cuGraphicsMapResources
CUGL	CUGRAPHICS, 232
cuGLCtxCreate, 196	CUgraphicsRegisterFlags
cuGraphicsGLRegisterBuffer, 197	CUDA_TYPES, 243
cuGraphicsGLRegisterImage, 197	CUgraphicsRegisterFlags_enum
cuWGLGetDevice, 198	CUDA_TYPES, 248
CUGL_DEPRECATED	cuGraphicsResourceGetMappedPointer
cuGLInit, 199	CUGRAPHICS, 233
cuGLMapBufferObject, 200	cuGraphicsResourceSetMapFlags
cuGLMapBufferObjectAsync, 200	CUGRAPHICS, 233
cuGLRegisterBufferObject, 201	cuGraphicsSubResourceGetMappedArray
cuGLSetBufferObjectMapFlags, 201	CUGRAPHICS, 234
cuGLUnmapBufferObject, 202	cuGraphicsUnmapResources
cuGLUnmapBufferObjectAsync, 203	CUGRAPHICS, 235
cuGLUnregisterBufferObject, 203	cuGraphicsUnregisterResource
cuGLCtxCreate	CUGRAPHICS, 235
CUGL, 196	CUINIT
cuGLInit	cuInit, 119
CUGL_DEPRECATED, 199	cuInit
cuGLMapBufferObject	CUINIT, 119
CUGL_DEPRECATED, 200	CUjit_fallback
cuGLMapBufferObjectAsync	CUDA_TYPES, 244
CUGL_DEPRECATED, 200	CUjit_fallback_enum
cuGLRegisterBufferObject	CUDA_TYPES, 248
CUGL DEPRECATED, 201	CUjit_option
cuGLSetBufferObjectMapFlags	CUDA_TYPES, 244
CUGL_DEPRECATED, 201	CUjit_option_enum
cuGLUnmapBufferObject	CUDA_TYPES, 248
CUGL_DEPRECATED, 202	
	CUjit_target CUDA_TYPES, 244
cuGLUnmapBufferObjectAsync	
CUGL_DEPRECATED, 203	CUjit_target_enum
cuGLUnregisterBufferObject	CUDA_TYPES, 249
CUGL_DEPRECATED, 203	cuLaunch
CUGRAPHICS	CUEXEC, 147
cuGraphicsMapResources, 232	cuLaunchGrid
cuGraphicsResourceGetMappedPointer, 233	CUEXEC, 147
cuGraphicsResourceSetMapFlags, 233	cuLaunchGridAsync
cuGraphicsSubResourceGetMappedArray, 234	CUEXEC, 148
cuGraphicsUnmapResources, 235	CUMEM
cuGraphicsUnregisterResource, 235	cuArray3DCreate, 154
cuGraphicsD3D10RegisterResource	cuArray3DGetDescriptor, 156
CUD3D10, 218	cuArrayCreate, 156
cuGraphicsD3D11RegisterResource	cuArrayDestroy, 158
CUD3D11, 230	cuArrayGetDescriptor, 158
cuGraphicsD3D9RegisterResource	cuMemAlloc, 159

cuMemAllocHost, 160	CUMEM, 173
cuMemAllocPitch, 160	cuMemcpyAtoHAsync
cuMemcpy2D, 161	CUMEM, 174
cuMemcpy2DAsync, 163	cuMemcpyDtoA
cuMemcpy2DUnaligned, 165	CUMEM, 174
cuMemcpy3D, 167	cuMemcpyDtoD
cuMemcpy3DAsync, 170	CUMEM, 175
cuMemcpyAtoA, 172	cuMemcpyDtoDAsync
cuMemcpyAtoD, 173	CUMEM, 176
cuMemcpyAtoH, 173	cuMemcpyDtoH
cuMemcpyAtoHAsync, 174	CUMEM, 176
cuMemcpyDtoA, 174	cuMemcpyDtoHAsync
cuMemcpyDtoD, 175	CUMEM, 177
cuMemcpyDtoDAsync, 176	cuMemcpyHtoA
cuMemcpyDtoH, 176	CUMEM, 177
cuMemcpyDtoHAsync, 177	cuMemcpyHtoAAsync
cuMemcpyHtoA, 177	CUMEM, 178
cuMemcpyHtoAAsync, 178	cuMemcpyHtoD
cuMemcpyHtoD, 179	CUMEM, 179
cuMemcpyHtoDAsync, 179	cuMemcpyHtoDAsync
cuMemFree, 180	CUMEM, 179
cuMemFreeHost, 180	cuMemFree
cuMemGetAddressRange, 181	CUMEM, 180
cuMemGetInfo, 181	cuMemFreeHost
cuMemHostAlloc, 182	CUMEM, 180
cuMemHostGetDevicePointer, 183	cuMemGetAddressRange
cuMemHostGetFlags, 184	CUMEM, 181
cuMemsetD16, 184	cuMemGetInfo
cuMemsetD2D16, 185	CUMEM, 181
cuMemsetD2D32, 185	cuMemHostAlloc
cuMemsetD2D8, 186	CUMEM, 182
cuMemsetD32, 186	cuMemHostGetDevicePointer
cuMemsetD8, 187	CUMEM, 183
cuMemAlloc	cuMemHostGetFlags
CUMEM, 159	CUMEM, 184
cuMemAllocHost	CUmemorytype
CUMEM, 160	CUDA_TYPES, 244
cuMemAllocPitch	CUmemorytype_enum
CUMEM, 160	CUDA_TYPES, 250
cuMemcpy2D	cuMemsetD16
CUMEM, 161	CUMEM, 184
cuMemcpy2DAsync	cuMemsetD2D16
CUMEM, 163	CUMEM, 185
cuMemcpy2DUnaligned	cuMemsetD2D32
CUMEM, 165	CUMEM, 185
cuMemcpy3D	cuMemsetD2D8
CUMEM, 167	CUMEM, 186
cuMemcpy3DAsync	cuMemsetD32
CUMEM, 170	CUMEM, 186
cuMemcpyAtoA	cuMemsetD8
CUMEM, 172	CUMEM, 187
cuMemcpyAtoD	CUMODULE
CUMEM, 173	cuModuleGetFunction, 132
cuMemcpyAtoH	cuModuleGetGlobal, 133
	Tur. 13 daile Get G10 dai, 133

cuModuleGetTexRef, 133	cuTexRefGetFlags, 191
cuModuleLoad, 134	cuTexRefGetFormat, 191
cuModuleLoadData, 134	cuTexRefSetAddress, 192
cuModuleLoadDataEx, 135	cuTexRefSetAddress2D, 192
cuModuleLoadFatBinary, 136	cuTexRefSetAddressMode, 193
cuModuleUnload, 136	cuTexRefSetArray, 193
cuModuleGetFunction	cuTexRefSetFilterMode, 194
CUMODULE, 132	cuTexRefSetFlags, 194
cuModuleGetGlobal	cuTexRefSetFormat, 195
CUMODULE, 133	cuTexRefCreate
cuModuleGetTexRef	CUTEXREF, 189
CUMODULE, 133	cuTexRefDestroy
cuModuleLoad	CUTEXREF, 189
CUMODULE, 134	cuTexRefGetAddress
cuModuleLoadData	CUTEXREF, 189
CUMODULE, 134	cuTexRefGetAddressMode
cuModuleLoadDataEx	CUTEXREF, 190
CUMODULE, 135	cuTexRefGetArray
cuModuleLoadFatBinary	CUTEXREF, 190
CUMODULE, 136	cuTexRefGetFilterMode
cuModuleUnload	CUTEXREF, 190
CUMODULE, 136	cuTexRefGetFlags
cuParamSetf	CUTEXREF, 191
CUEXEC, 148	cuTexRefGetFormat
cuParamSeti	CUTEXREF, 191
CUEXEC, 149	cuTexRefSetAddress
cuParamSetSize	CUTEXREF, 192
CUEXEC, 149	cuTexRefSetAddress2D
cuParamSetTexRef	CUTEXREF, 192
CUEXEC, 150	cuTexRefSetAddressMode
cuParamSetv	CUTEXREF, 193
CUEXEC, 150	cuTexRefSetArray
CUresult	CUTEXREF, 193
CUDA_TYPES, 244	cuTexRefSetFilterMode
CUSTREAM	CUTEXREF, 194
cuStreamCreate, 138	cuTexRefSetFlags
cuStreamDestroy, 138	CUTEXREF, 194
cuStreamQuery, 139	cuTexRefSetFormat
cuStreamSynchronize, 139	CUTEXREF, 195
cuStreamCreate	CUVERSION
CUSTREAM, 138	cuDriverGetVersion, 126
cuStreamDestroy	cuWGLGetDevice
CUSTREAM, 138	CUGL, 198
cuStreamQuery	0002, 170
CUSTREAM, 139	Data types used by CUDA driver, 237
cuStreamSynchronize	Data types used by CUDA Runtime, 112
CUSTREAM, 139	Device Management, 13, 120
CUTEXREF	Direct3D 10 Interoperability, 79, 217
cuTexRefCreate, 189	Direct3D 11 Interoperability, 90, 229
cuTexRefDestroy, 189	Direct3D 9 Interoperability, 67, 205
cuTexRefGetAddress, 189	2.100002 / Interoperating, 07, 200
cuTexRefGetAddressMode, 190	Error Handling, 11
cuTexRefGetArray, 190	Event Management, 21, 140
cuTexRefGetFilterMode, 190	Execution Control, 25, 144
outentered mentione, 170	2.100auon Conuoi, 20, 1-17

```
Graphics Interoperability, 93, 232

Initialization, 119

make_cudaExtent
    CUDART_MEMORY, 57

make_cudaPitchedPtr
    CUDART_MEMORY, 57

make_cudaPos
    CUDART_MEMORY, 57

Memory Management, 29, 152

Module Management, 132

OpenGL Interoperability, 59, 196

Stream Management, 19, 138

Texture Reference Management, 97, 188

Thread Management, 10

Version Management, 102, 126
```

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, GeForce, Tesla, and Quadro are trademarks or registered trademarks of NVIDIA Corporation. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2010 NVIDIA Corporation. All rights reserved.



2701 San Tomas Expressway Santa Clara, CA 95050 www.nvidia.com