

GOOPAX Reference

3.3.0

Generated by Doxygen 1.8.14

Contents

- 1 Module Index** **1**
- 1.1 Modules 1

- 2 Hierarchical Index** **3**
- 2.1 Class Hierarchy 3

- 3 Class Index** **5**
- 3.1 Class List 5

- 4 Module Documentation** **7**
- 4.1 atomic operations 7
- 4.1.1 Detailed Description 7
- 4.1.2 Function Documentation 8
- 4.1.2.1 atomic_add() 8
- 4.1.2.2 atomic_and() 8
- 4.1.2.3 atomic_cmpxchg() [1/4] 8
- 4.1.2.4 atomic_cmpxchg() [2/4] 9
- 4.1.2.5 atomic_cmpxchg() [3/4] 10
- 4.1.2.6 atomic_cmpxchg() [4/4] 11
- 4.1.2.7 atomic_load() [1/4] 12
- 4.1.2.8 atomic_load() [2/4] 12
- 4.1.2.9 atomic_load() [3/4] 12
- 4.1.2.10 atomic_load() [4/4] 13
- 4.1.2.11 atomic_max() 13
- 4.1.2.12 atomic_min() 14

4.1.2.13	<code>atomic_or()</code>	14
4.1.2.14	<code>atomic_sub()</code> [1/4]	14
4.1.2.15	<code>atomic_sub()</code> [2/4]	16
4.1.2.16	<code>atomic_sub()</code> [3/4]	16
4.1.2.17	<code>atomic_sub()</code> [4/4]	17
4.1.2.18	<code>atomic_xchg()</code> [1/3]	17
4.1.2.19	<code>atomic_xchg()</code> [2/3]	18
4.1.2.20	<code>atomic_xchg()</code> [3/3]	18
4.1.2.21	<code>atomic_xor()</code>	18
4.2	memory resources	20
4.2.1	Detailed Description	20
4.2.2	Typedef Documentation	20
4.2.2.1	<code>local_mem</code>	20
4.2.2.2	<code>private_mem</code>	20
4.2.3	Enumeration Type Documentation	20
4.2.3.1	<code>BUFFER_FLAGS</code>	20
4.3	OpenCL interoperability	22
4.3.1	Detailed Description	22
4.3.2	Function Documentation	22
4.3.2.1	<code>create_from_cl()</code> [1/3]	22
4.3.2.2	<code>create_from_cl()</code> [2/3]	22
4.3.2.3	<code>create_from_cl()</code> [3/3]	24
4.3.2.4	<code>get_cl_context()</code>	24
4.3.2.5	<code>get_cl_device()</code>	24
4.3.2.6	<code>get_cl_device_extensions()</code>	24
4.3.2.7	<code>get_cl_mem()</code>	24
4.3.2.8	<code>get_cl_platform()</code>	25
4.3.2.9	<code>get_cl_platform_extensions()</code>	25
4.3.2.10	<code>get_cl_queue()</code>	25
4.3.2.11	<code>get_device_from_cl_queue()</code>	25

4.3.2.12	<code>have_cl_device_extension()</code>	25
4.3.2.13	<code>have_cl_platform_extension()</code>	25
4.4	cpu and gpu functions	26
4.4.1	Detailed Description	27
4.4.2	Function Documentation	27
4.4.2.1	<code>abs()</code>	27
4.4.2.2	<code>acospi()</code> [1/3]	27
4.4.2.3	<code>acospi()</code> [2/3]	27
4.4.2.4	<code>acospi()</code> [3/3]	27
4.4.2.5	<code>asinpi()</code> [1/3]	28
4.4.2.6	<code>asinpi()</code> [2/3]	28
4.4.2.7	<code>asinpi()</code> [3/3]	28
4.4.2.8	<code>atanpi()</code> [1/3]	28
4.4.2.9	<code>atanpi()</code> [2/3]	28
4.4.2.10	<code>atanpi()</code> [3/3]	28
4.4.2.11	<code>clamp()</code> [1/2]	28
4.4.2.12	<code>clamp()</code> [2/2]	29
4.4.2.13	<code>clz()</code> [1/3]	29
4.4.2.14	<code>clz()</code> [2/3]	29
4.4.2.15	<code>clz()</code> [3/3]	29
4.4.2.16	<code>cospi()</code> [1/3]	29
4.4.2.17	<code>cospi()</code> [2/3]	29
4.4.2.18	<code>cospi()</code> [3/3]	29
4.4.2.19	<code>max()</code> [1/3]	30
4.4.2.20	<code>max()</code> [2/3]	30
4.4.2.21	<code>max()</code> [3/3]	30
4.4.2.22	<code>min()</code> [1/3]	30
4.4.2.23	<code>min()</code> [2/3]	30
4.4.2.24	<code>min()</code> [3/3]	30
4.4.2.25	<code>popcount()</code> [1/2]	31

4.4.2.26	<code>popcount()</code> [2/2]	31
4.4.2.27	<code>rotl()</code> [1/2]	31
4.4.2.28	<code>rotl()</code> [2/2]	31
4.4.2.29	<code>rotr()</code> [1/2]	31
4.4.2.30	<code>rotr()</code> [2/2]	31
4.4.2.31	<code>sinpi()</code> [1/3]	32
4.4.2.32	<code>sinpi()</code> [2/3]	32
4.4.2.33	<code>sinpi()</code> [3/3]	32
4.4.2.34	<code>tanpi()</code> [1/3]	32
4.4.2.35	<code>tanpi()</code> [2/3]	32
4.4.2.36	<code>tanpi()</code> [3/3]	32
4.5	CUDA interoperability	33
4.5.1	Detailed Description	33
4.5.2	Function Documentation	33
4.5.2.1	<code>create_from_cuda()</code>	33
4.5.2.2	<code>get_cuda_device()</code>	33
4.6	Thread numbers	34
4.6.1	Detailed Description	34
4.6.2	Function Documentation	34
4.6.2.1	<code>global_id()</code>	34
4.6.2.2	<code>global_size()</code>	35
4.6.2.3	<code>group_id()</code>	35
4.6.2.4	<code>local_id()</code>	35
4.6.2.5	<code>local_size()</code>	35
4.6.2.6	<code>max_local_mem()</code>	35
4.6.2.7	<code>num_groups()</code>	35
4.6.2.8	<code>num_subthreads()</code>	35
4.6.2.9	<code>thread_id_in_warp()</code>	36
4.6.2.10	<code>warp_id()</code>	36
4.6.2.11	<code>warp_id_in_group()</code>	36

4.6.2.12	warp_size()	36
4.7	Device	37
4.7.1	Detailed Description	37
4.7.2	Enumeration Type Documentation	37
4.7.2.1	envmode	37
4.7.3	Function Documentation	37
4.7.3.1	default_device()	38
4.7.3.2	devices()	38
4.7.3.3	get_current_build_device()	38
4.7.3.4	init_params()	38
4.7.3.5	operator" ()	40
4.8	Gather return values	41
4.8.1	Detailed Description	41
4.9	OpenGL interoperability	42
4.9.1	Detailed Description	42
4.9.2	Function Documentation	42
4.9.2.1	create_from_gl() [1/3]	42
4.9.2.2	create_from_gl() [2/3]	42
4.9.2.3	create_from_gl() [3/3]	43
4.9.2.4	flush_gl_interop()	43
4.10	Math	44
4.10.1	Detailed Description	44
4.10.2	Function Documentation	44
4.10.2.1	pow() [1/4]	44
4.10.2.2	pow() [2/4]	44
4.10.2.3	pow() [3/4]	45
4.10.2.4	pow() [4/4]	45
4.11	debugging	46
4.11.1	Detailed Description	46
4.11.2	Macro Definition Documentation	46

4.11.2.1	gpu_assert	46
4.12	Types	47
4.12.1	Detailed Description	47
4.13	Operators	48
4.13.1	Detailed Description	48
4.13.2	Function Documentation	48
4.13.2.1	cond() [1/5]	48
4.13.2.2	cond() [2/5]	48
4.13.2.3	cond() [3/5]	49
4.13.2.4	cond() [4/5]	49
4.13.2.5	cond() [5/5]	49
4.14	Image	51
4.14.1	Detailed Description	51
4.14.2	Enumeration Type Documentation	51
4.14.2.1	IMAGE_FLAGS	51
4.15	Kernel	52
4.15.1	Detailed Description	52
4.16	Loops	53
4.16.1	Detailed Description	53
4.16.2	Macro Definition Documentation	53
4.16.2.1	gpu_else	54
4.16.2.2	gpu_elseif	54
4.16.2.3	gpu_for_global	54
4.16.2.4	gpu_for_group	54
4.16.2.5	gpu_for_local	54
4.16.2.6	gpu_if	54
4.16.2.7	gpu_while	54
4.17	operators	55
4.17.1	Detailed Description	55
4.18	Reinterpret	56

4.18.1 Detailed Description	56
4.18.2 Function Documentation	56
4.18.2.1 reinterpret() [1/2]	56
4.18.2.2 reinterpret() [2/2]	56
4.19 random numbers	58
4.19.1 Detailed Description	58
4.20 goopax struct types	59
4.20.1 Detailed Description	59
4.20.2 Function Documentation	59
4.20.2.1 output_goopax_struct()	59
4.21 Thread_communication	60
4.21.1 Detailed Description	60
4.21.2 Function Documentation	60
4.21.2.1 ballot()	60
4.21.2.2 shuffle()	61
4.21.2.3 work_group_all()	61
4.21.2.4 work_group_any()	62
4.21.2.5 work_group_broadcast()	62
4.21.2.6 work_group_reduce()	63
4.21.2.7 work_group_reduce_add() [1/2]	63
4.21.2.8 work_group_reduce_add() [2/2]	63
4.21.2.9 work_group_reduce_max() [1/2]	64
4.21.2.10 work_group_reduce_max() [2/2]	64
4.21.2.11 work_group_reduce_min() [1/2]	64
4.21.2.12 work_group_reduce_min() [2/2]	65
4.21.2.13 work_group_scan_exclusive_add()	65
4.21.2.14 work_group_scan_exclusive_max()	65
4.21.2.15 work_group_scan_exclusive_min()	66
4.21.2.16 work_group_scan_inclusive_add()	66
4.21.2.17 work_group_scan_inclusive_max()	66
4.21.2.18 work_group_scan_inclusive_min()	67

5 Class Documentation	69
5.1 <code>buffer< T, SIZE_TYPE ></code> Class Template Reference	69
5.1.1 Detailed Description	70
5.1.2 Constructor & Destructor Documentation	71
5.1.2.1 <code>buffer()</code> [1/5]	71
5.1.2.2 <code>buffer()</code> [2/5]	71
5.1.2.3 <code>buffer()</code> [3/5]	71
5.1.2.4 <code>buffer()</code> [4/5]	71
5.1.2.5 <code>buffer()</code> [5/5]	72
5.1.3 Member Function Documentation	72
5.1.3.1 <code>copy()</code> [1/2]	72
5.1.3.2 <code>copy()</code> [2/2]	72
5.1.3.3 <code>copy_from_host()</code> [1/2]	72
5.1.3.4 <code>copy_from_host()</code> [2/2]	73
5.1.3.5 <code>copy_to_host()</code> [1/2]	73
5.1.3.6 <code>copy_to_host()</code> [2/2]	73
5.1.3.7 <code>fill()</code> [1/2]	74
5.1.3.8 <code>fill()</code> [2/2]	74
5.1.3.9 <code>get_device()</code>	74
5.1.3.10 <code>max()</code> [1/2]	74
5.1.3.11 <code>max()</code> [2/2]	75
5.1.3.12 <code>min()</code> [1/2]	75
5.1.3.13 <code>min()</code> [2/2]	75
5.1.3.14 <code>operator=()</code> [1/2]	75
5.1.3.15 <code>operator=()</code> [2/2]	75
5.1.3.16 <code>size()</code>	75
5.1.3.17 <code>sum()</code> [1/2]	75
5.1.3.18 <code>sum()</code> [2/2]	76
5.1.3.19 <code>to_vector()</code>	76
5.1.4 Friends And Related Function Documentation	76

5.1.4.1	operator<<	76
5.1.4.2	reinterpret	76
5.2	buffer_map< T, is_const > Class Template Reference	76
5.2.1	Detailed Description	77
5.2.2	Constructor & Destructor Documentation	77
5.2.2.1	buffer_map() [1/4]	77
5.2.2.2	buffer_map() [2/4]	78
5.2.2.3	buffer_map() [3/4]	78
5.2.2.4	buffer_map() [4/4]	78
5.2.3	Member Function Documentation	79
5.2.3.1	begin() [1/2]	79
5.2.3.2	begin() [2/2]	79
5.2.3.3	data() [1/2]	79
5.2.3.4	data() [2/2]	79
5.2.3.5	end() [1/2]	79
5.2.3.6	end() [2/2]	79
5.2.3.7	operator[]() [1/2]	79
5.2.3.8	operator[]() [2/2]	80
5.3	full_kernel Struct Reference	80
5.3.1	Detailed Description	80
5.4	gather_add< T > Struct Template Reference	81
5.4.1	Detailed Description	81
5.4.2	Member Function Documentation	81
5.4.2.1	init()	81
5.4.2.2	op()	81
5.5	gather_max< T > Struct Template Reference	81
5.5.1	Detailed Description	81
5.6	gather_min< T > Struct Template Reference	82
5.6.1	Detailed Description	82
5.7	gettype< T, EIF > Struct Template Reference	82

5.7.1	Detailed Description	82
5.8	goopax_device Class Reference	82
5.8.1	Detailed Description	84
5.8.2	Constructor & Destructor Documentation	84
5.8.2.1	goopax_device()	84
5.8.3	Member Function Documentation	84
5.8.3.1	operator=()	84
5.9	EX::goopax_exception Struct Reference	85
5.9.1	Detailed Description	85
5.10	goopax_future< T > Class Template Reference	85
5.10.1	Detailed Description	85
5.10.2	Member Function Documentation	86
5.10.2.1	get()	86
5.11	goopax_future< void > Class Template Reference	86
5.11.1	Detailed Description	86
5.12	goopax_future_info Class Reference	87
5.12.1	Detailed Description	87
5.13	goopax_struct_type< std::array< T, N > > Struct Template Reference	87
5.13.1	Detailed Description	87
5.14	gpu_ostream Class Reference	87
5.14.1	Detailed Description	88
5.14.2	Constructor & Destructor Documentation	88
5.14.2.1	gpu_ostream()	88
5.15	gpu_type< T, scope > Class Template Reference	89
5.15.1	Detailed Description	91
5.15.2	Constructor & Destructor Documentation	91
5.15.2.1	gpu_type() [1/7]	91
5.15.2.2	gpu_type() [2/7]	91
5.15.2.3	gpu_type() [3/7]	91
5.15.2.4	gpu_type() [4/7]	92

5.15.2.5	gpu_type() [5/7]	92
5.15.2.6	gpu_type() [6/7]	92
5.15.2.7	gpu_type() [7/7]	92
5.15.3	Member Function Documentation	92
5.15.3.1	gpu_break()	92
5.15.3.2	gpu_continue()	92
5.15.3.3	operator+()	93
5.15.3.4	operator++() [1/2]	93
5.15.3.5	operator++() [2/2]	93
5.15.3.6	operator-() [1/2]	93
5.15.3.7	operator-() [2/2]	93
5.15.3.8	operator--() [1/2]	93
5.15.3.9	operator--() [2/2]	93
5.15.3.10	operator<=()	94
5.15.3.11	operator=()	94
5.15.3.12	operator>()	94
5.16	image_array_buffer< DIM, T, normalized > Class Template Reference	94
5.16.1	Detailed Description	95
5.16.2	Constructor & Destructor Documentation	95
5.16.2.1	image_array_buffer() [1/2]	95
5.16.2.2	image_array_buffer() [2/2]	95
5.16.3	Member Function Documentation	96
5.16.3.1	array_size()	96
5.16.3.2	copy() [1/2]	96
5.16.3.3	copy() [2/2]	96
5.16.3.4	copy_from_host()	97
5.16.3.5	copy_to_host() [1/2]	97
5.16.3.6	copy_to_host() [2/2]	97
5.16.3.7	fill() [1/2]	98
5.16.3.8	fill() [2/2]	98

5.16.3.9	operator[]() [1/2]	98
5.16.3.10	operator[]() [2/2]	98
5.16.3.11	to_vector()	99
5.17	image_array_buffer_map< DIM, T, is_const > Class Template Reference	99
5.17.1	Detailed Description	99
5.17.2	Constructor & Destructor Documentation	99
5.17.2.1	image_array_buffer_map() [1/4]	99
5.17.2.2	image_array_buffer_map() [2/4]	100
5.17.2.3	image_array_buffer_map() [3/4]	100
5.17.2.4	image_array_buffer_map() [4/4]	100
5.17.3	Member Function Documentation	101
5.17.3.1	array_size()	101
5.17.3.2	dimensions()	101
5.17.3.3	operator[]() [1/2]	101
5.17.3.4	operator[]() [2/2]	101
5.18	image_array_resource< DIM, T, normalized > Class Template Reference	102
5.18.1	Detailed Description	102
5.18.2	Constructor & Destructor Documentation	102
5.18.2.1	image_array_resource()	102
5.18.3	Member Function Documentation	103
5.18.3.1	operator[]() [1/2]	103
5.18.3.2	operator[]() [2/2]	103
5.19	image_buffer< DIM, T, normalized > Class Template Reference	103
5.19.1	Detailed Description	104
5.19.2	Constructor & Destructor Documentation	105
5.19.2.1	image_buffer() [1/2]	105
5.19.2.2	image_buffer() [2/2]	105
5.19.3	Member Function Documentation	105
5.19.3.1	copy() [1/2]	105
5.19.3.2	copy() [2/2]	106

5.19.3.3	copy_from_host() [1/2]	106
5.19.3.4	copy_from_host() [2/2]	106
5.19.3.5	copy_to_host()	107
5.19.3.6	fill() [1/2]	107
5.19.3.7	fill() [2/2]	107
5.19.3.8	to_vector()	108
5.20	image_buffer_map< DIM, T, is_const > Class Template Reference	108
5.20.1	Detailed Description	108
5.20.2	Constructor & Destructor Documentation	108
5.20.2.1	image_buffer_map() [1/4]	108
5.20.2.2	image_buffer_map() [2/4]	109
5.20.2.3	image_buffer_map() [3/4]	109
5.20.2.4	image_buffer_map() [4/4]	109
5.20.3	Member Function Documentation	110
5.20.3.1	dimensions()	110
5.20.3.2	operator[]() [1/2]	110
5.20.3.3	operator[]() [2/2]	110
5.21	image_resource< DIM, T, normalized > Class Template Reference	110
5.21.1	Detailed Description	111
5.21.2	Constructor & Destructor Documentation	111
5.21.2.1	image_resource()	111
5.21.3	Member Function Documentation	112
5.21.3.1	read() [1/4]	112
5.21.3.2	read() [2/4]	112
5.21.3.3	read() [3/4]	112
5.21.3.4	read() [4/4]	113
5.21.3.5	write() [1/2]	113
5.21.3.6	write() [2/2]	113
5.22	image_array_buffer< DIM, T, normalized >::image_slice< is_const > Struct Template Reference	114
5.22.1	Detailed Description	114

5.22.2	Member Function Documentation	114
5.22.2.1	copy() [1/2]	114
5.22.2.2	copy() [2/2]	115
5.22.2.3	copy_from_host() [1/2]	115
5.22.2.4	copy_from_host() [2/2]	115
5.22.2.5	copy_to_host() [1/2]	115
5.22.2.6	copy_to_host() [2/2]	116
5.22.2.7	dimensions()	116
5.22.2.8	to_vector()	116
5.23	image_array_resource< DIM, T, normalized >::image_slice< RESREF > Struct Template Reference	116
5.23.1	Detailed Description	117
5.23.2	Member Function Documentation	117
5.23.2.1	read() [1/4]	117
5.23.2.2	read() [2/4]	117
5.23.2.3	read() [3/4]	117
5.23.2.4	read() [4/4]	119
5.23.2.5	write() [1/2]	119
5.23.2.6	write() [2/2]	119
5.24	kernel< K > Struct Template Reference	120
5.24.1	Detailed Description	120
5.24.2	Constructor & Destructor Documentation	121
5.24.2.1	kernel()	121
5.24.3	Member Function Documentation	121
5.24.3.1	operator>()	121
5.24.3.2	operator=()	121
5.25	local_scope_mem< T, SCOPE > Class Template Reference	121
5.25.1	Detailed Description	122
5.25.2	Constructor & Destructor Documentation	122
5.25.2.1	local_scope_mem()	122
5.25.3	Member Function Documentation	123

5.25.3.1	begin() [1/2]	123
5.25.3.2	begin() [2/2]	123
5.25.3.3	copy() [1/4]	123
5.25.3.4	copy() [2/4]	123
5.25.3.5	copy() [3/4]	124
5.25.3.6	copy() [4/4]	124
5.25.3.7	end() [1/2]	124
5.25.3.8	end() [2/2]	124
5.25.3.9	fill()	125
5.25.3.10	size()	125
5.26	numeric_limits< goopax::gpu_type< T > > Struct Template Reference	125
5.26.1	Detailed Description	125
5.27	resource< T, SIZE_TYPE, is_const > Class Template Reference	125
5.27.1	Detailed Description	126
5.27.2	Constructor & Destructor Documentation	126
5.27.2.1	resource() [1/3]	126
5.27.2.2	resource() [2/3]	127
5.27.2.3	resource() [3/3]	127
5.27.3	Member Function Documentation	127
5.27.3.1	begin() [1/2]	127
5.27.3.2	begin() [2/2]	127
5.27.3.3	copy()	128
5.27.3.4	end() [1/2]	128
5.27.3.5	end() [2/2]	128
5.28	sresource< STR, BASE > Struct Template Reference	128
5.28.1	Detailed Description	128
5.29	WELL512 Class Reference	129
5.29.1	Detailed Description	129
5.29.2	Member Function Documentation	129
5.29.2.1	seed()	129
5.30	WELL512lib Class Reference	129
5.30.1	Detailed Description	129
5.30.2	Constructor & Destructor Documentation	129
5.30.2.1	WELL512lib()	130
5.30.3	Member Function Documentation	130
5.30.3.1	gen_vec()	130

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

- atomic operations 7
- memory resources 20
- OpenCL interoperability 22
- cpu and gpu functions 26
- CUDA interoperability 33
- Thread numbers 34
- Device 37
- Gather return values 41
- OpenGL interoperability 42
- Math 44
- debugging 46
- Types 47
- Operators 48
- Image 51
- Kernel 52
- Loops 53
- operators 55
- Reinterpret 56
- random numbers 58
- goopax struct types 59
- Thread_communication 60

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

buffer< T, SIZE_TYPE >	69
buffer_map< T, is_const >	76
full_kernel	80
kernel< K >	120
kernel< auto_kernel< FUNC, ReturnType(ClassType::*)(Args...) const > >	120
kernel< auto_kernel< FUNC, void(ClassType::*)(Args...) const > >	120
kernel< gather_prog< DEST, SRC, burstsize > >	120
kernel< minmaxsum_prog< OP, size_type > >	120
gather_add< T >	81
gather_max< T >	81
gather_min< T >	82
gettype< T, EIF >	82
goopax_device	82
EX::goopax_exception	85
goopax_future_info	87
goopax_future< T >	85
goopax_future< void >	86
goopax_struct_type< std::array< T, N > >	87
gpu_ostream	87
gpu_type< T, scope >	89
image_array_buffer< DIM, T, normalized >	94
image_array_buffer_map< DIM, T, is_const >	99
image_array_resource< DIM, T, normalized >	102
image_buffer< DIM, T, normalized >	103
image_buffer_map< DIM, T, is_const >	108
image_resource< DIM, T, normalized >	110
image_array_buffer< DIM, T, normalized >::image_slice< is_const >	114
image_array_resource< DIM, T, normalized >::image_slice< RESREF >	116
local_scope_mem< T, SCOPE >	121
numeric_limits< goopax::gpu_type< T > >	125
resource< T, SIZE_TYPE, is_const >	125
sresource< STR, BASE >	128
WELL512	129
WELL512lib	129

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

buffer< T, SIZE_TYPE >	69
buffer_map< T, is_const >	76
full_kernel	80
gather_add< T >	81
gather_max< T >	81
gather_min< T >	82
gettype< T, EIF >	
Type info and type conversion mechanism With the gettype struct, CPU and GPU types can be converted to each other, and type information obtained typename gettype<T>::cpu converts T to the corresponding CPU type. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. typename gettype<T>::gpu converts T to the corresponding GPU type. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. gettype<T>::size provides the size of type T in bytes. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. typename gettype<T>::template change<U>::type changes the type of T to type U while preserving CPU/GPU/debug mode. If T is CPU type, the resulting type is also a CPU type. If T is GPU type, the resulting type is a GPU type. If T is debug data type, the resulting type is also a debug data type	
goopax_device	82
Goopax device. This class is used to reference the physical devices in the system	
EX::goopax_exception	
Base exception thrown by goopax	
goopax_future< T >	85
goopax_future< void >	86
goopax_future_info	87
goopax_struct_type< std::array< T, N > >	
Std::array and derived classes	
gpu_ostream	87
gpu_type< T, scope >	89
image_array_buffer< DIM, T, normalized >	94
image_array_buffer_map< DIM, T, is_const >	99
image_array_resource< DIM, T, normalized >	102
image_buffer< DIM, T, normalized >	103
image_buffer_map< DIM, T, is_const >	108
image_resource< DIM, T, normalized >	110
image_array_buffer< DIM, T, normalized >::image_slice< is_const >	
Reference to single image in the image array	
	114

image_array_resource< DIM, T, normalized >::image_slice< RESREF >	
Reference to single image in the image array	116
kernel< K >	
Kernel class to derive gpu kernels from	120
local_scope_mem< T, SCOPE >	121
numeric_limits< goopax::gpu_type< T > >	
Numeric_limits overload for GPU types	125
resource< T, SIZE_TYPE, is_const >	125
sresource< STR, BASE >	128
WELL512	129
WELL512lib	129

Chapter 4

Module Documentation

4.1 atomic operations

Functions

- `gpu_uint32 atomic_and (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_uint32 atomic_or (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_uint32 atomic_xor (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_uint32 atomic_min (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_uint32 atomic_max (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_uint32 atomic_xchg (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_float atomic_xchg (gpu_float &ref, const gpu_float value)`
- `gpu_double atomic_xchg (gpu_double &ref, const gpu_double value)`
- `gpu_uint32 atomic_add (gpu_uint32 &ref, const gpu_uint32 value)`
- `gpu_uint32 atomic_load (const gpu_uint32 &ref)`
- `gpu_int32 atomic_load (const gpu_int32 &ref)`
- `gpu_uint64 atomic_load (const gpu_uint64 &ref)`
- `gpu_int64 atomic_load (const gpu_int64 &ref)`
- `gpu_int32 atomic_sub (gpu_int32 &ref, gpu_int32 value)`
- `gpu_int32 atomic_cmpxchg (gpu_int32 &ref, gpu_int32 cmp, gpu_int32 value)`
- `gpu_uint32 atomic_sub (gpu_uint32 &ref, gpu_uint32 value)`
- `gpu_uint32 atomic_cmpxchg (gpu_uint32 &ref, gpu_uint32 cmp, gpu_uint32 value)`
- `gpu_int64 atomic_sub (gpu_int64 &ref, gpu_int64 value)`
- `gpu_int64 atomic_cmpxchg (gpu_int64 &ref, gpu_int64 cmp, gpu_int64 value)`
- `gpu_uint64 atomic_sub (gpu_uint64 &ref, gpu_uint64 value)`
- `gpu_uint64 atomic_cmpxchg (gpu_uint64 &ref, gpu_uint64 cmp, gpu_uint64 value)`

4.1.1 Detailed Description

atomic functions are used to atomically modify values in global or local memory.

example:

```
struct foo :
kernel<foo>
{
    void program(resource<int>& a, resource<pair<uint64_t, double>>& B)
    {
        atomic_add(a[0], 1); // increments a[0] by 1
        gpu_double x = atomic_xchg(b[12].second, 3.14); // Writes 3.14 at B[12].second,
        returns in x the previous value
    }
};
```

4.1.2 Function Documentation

4.1.2.1 atomic_add()

```
gpu_uint32 goopax::impl::atomic_add (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic add

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>value</i>	value to add

Returns

previous value

4.1.2.2 atomic_and()

```
gpu_uint32 goopax::impl::atomic_and (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic and

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>value</i>	value to perform and operation with

Returns

previous value

4.1.2.3 atomic_cmpxchg() [1/4]

```
gpu_int32 goopax::impl::atomic_cmpxchg (
    gpu_int32 & ref,
```

```
gpu_int32 cmp,  
gpu_int32 value ) [inline]
```

atomic compare and exchange

If `ref == cmp`, write `ref = value`. Otherwise do nothing.

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>cmp</i>	value to compare with
<i>value</i>	value to write

Returns

previous value

4.1.2.4 atomic_cmpxchg() [2/4]

```
gpu_uint32 goopax::impl::atomic_cmpxchg (  
    gpu_uint32 & ref,  
    gpu_uint32 cmp,  
    gpu_uint32 value ) [inline]
```

atomic compare and exchange

If `ref == cmp`, write `ref = value`. Otherwise do nothing.

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>cmp</i>	value to compare with
<i>value</i>	value to write

Returns

previous value

4.1.2.5 `atomic_cmpxchg()` [3/4]

```
gpu_int64 goopax::impl::atomic_cmpxchg (  
    gpu_int64 & ref,  
    gpu_int64 cmp,  
    gpu_int64 value ) [inline]
```

atomic compare and exchange

If `ref == cmp`, write `ref = value`. Otherwise do nothing.

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>cmp</i>	value to compare with
<i>value</i>	value to write

Returns

previous value

4.1.2.6 atomic_cmpxchg() [4/4]

```
gpu_uint64 goopax::impl::atomic_cmpxchg (
    gpu_uint64 & ref,
    gpu_uint64 cmp,
    gpu_uint64 value ) [inline]
```

atomic compare and exchange

If `ref == cmp`, write `ref = value`. Otherwise do nothing.

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>cmp</i>	value to compare with
<i>value</i>	value to write
Generated by Doxygen	

Returns

previous value

4.1.2.7 atomic_load() [1/4]

```
gpu_uint32 goopax::impl::atomic_load (
    const gpu_uint32 & ref ) [inline]
```

atomic load

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
------------	---

Returns

value at location 'ref'

4.1.2.8 atomic_load() [2/4]

```
gpu_int32 goopax::impl::atomic_load (
    const gpu_int32 & ref ) [inline]
```

atomic load

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
------------	---

Returns

value at location 'ref'

4.1.2.9 atomic_load() [3/4]

```
gpu_uint64 goopax::impl::atomic_load (
    const gpu_uint64 & ref ) [inline]
```

atomic load

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
------------	---

Returns

value at location 'ref'

4.1.2.10 atomic_load() [4/4]

```
gpu_int64 goopax::impl::atomic_load (
    const gpu_int64 & ref ) [inline]
```

atomic load

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
------------	---

Returns

value at location 'ref'

4.1.2.11 atomic_max()

```
gpu_uint32 goopax::impl::atomic_max (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic max

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>value</i>	value to perform max operation with

Returns

previous value

4.1.2.12 atomic_min()

```
gpu_uint32 goopax::impl::atomic_min (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic min

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>value</i>	value to perform min operation with

Returns

previous value

4.1.2.13 atomic_or()

```
gpu_uint32 goopax::impl::atomic_or (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic or

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>value</i>	value to perform or operation with

Returns

previous value

4.1.2.14 atomic_sub() [1/4]

```
gpu_int32 goopax::impl::atomic_sub (
    gpu_int32 & ref,
    gpu_int32 value ) [inline]
```

atomic sub

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
------------	---

Returns

previous value

4.1.2.15 atomic_sub() [2/4]

```
gpu_uint32 goopax::impl::atomic_sub (  
    gpu_uint32 & ref,  
    gpu_uint32 value ) [inline]
```

atomic sub

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
------------	---

Returns

previous value

4.1.2.16 atomic_sub() [3/4]

```
gpu_int64 goopax::impl::atomic_sub (  
    gpu_int64 & ref,  
    gpu_int64 value ) [inline]
```

atomic sub

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
------------	---

Returns

previous value

4.1.2.17 `atomic_sub()` [4/4]

```
gpu_uint64 goopax::impl::atomic_sub (
    gpu_uint64 & ref,
    gpu_uint64 value ) [inline]
```

atomic sub

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
------------	---

Returns

previous value

4.1.2.18 `atomic_xchg()` [1/3]

```
gpu_uint32 goopax::impl::atomic_xchg (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic exchange

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
<i>value</i>	value to write

Returns

previous value

4.1.2.19 atomic_xchg() [2/3]

```
gpu_float goopax::impl::atomic_xchg (
    gpu_float & ref,
    const gpu_float value ) [inline]
```

atomic exchange

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
<i>value</i>	value to write

Returns

previous value

4.1.2.20 atomic_xchg() [3/3]

```
gpu_double goopax::impl::atomic_xchg (
    gpu_double & ref,
    const gpu_double value ) [inline]
```

atomic exchange

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral or floating point type.
<i>value</i>	value to write

Returns

previous value

4.1.2.21 atomic_xor()

```
gpu_uint32 goopax::impl::atomic_xor (
    gpu_uint32 & ref,
    const gpu_uint32 value ) [inline]
```

atomic xor

Parameters

<i>ref</i>	reference to local or global memory address. The value referenced must be of 32 or 64 bit signed or unsigned integral type.
<i>value</i>	value to perform xor operation with

Returns

previous value

4.2 memory resources

Classes

- class `resource< T, SIZE_TYPE, is_const >`
- class `buffer_map< T, is_const >`
- class `buffer< T, SIZE_TYPE >`
- class `local_scope_mem< T, SCOPE >`
- struct `sresource< STR, BASE >`

Typedefs

- template<typename T >
using `local_mem` = `local_scope_mem< T, memory::threadgroup >`
- template<typename T >
using `private_mem` = `local_scope_mem< T, memory::thread >`

Enumerations

- enum `BUFFER_FLAGS` { `BUFFER_READ_WRITE` = 1, `BUFFER_READ_ONLY` = 2, `BUFFER_WRITE_ONLY` = 4, `BUFFER_WRITE_DISCARD` = 8 }

4.2.1 Detailed Description

4.2.2 Typedef Documentation

4.2.2.1 local_mem

```
using local_mem = local_scope_mem<T, memory::threadgroup>
```

local memory. This memory can be accessed by all threads in the workgroup.

4.2.2.2 private_mem

```
using private_mem = local_scope_mem<T, memory::thread>
```

private memory. Each thread has its own private memory. No other thread can access it.

4.2.3 Enumeration Type Documentation

4.2.3.1 BUFFER_FLAGS

```
enum BUFFER_FLAGS
```

Buffer access flags

Enumerator

BUFFER_READ_WRITE	read/write access
BUFFER_READ_ONLY	read-only access
BUFFER_WRITE_ONLY	write-only access
BUFFER_WRITE_DISCARD	write-only, previous values are not preserved

4.3 OpenCL interoperability

Functions

- static `buffer create_from_cl` (`goopax_device` device, `_cl_mem` *mem)
- `cl_platform_id get_cl_platform` (`goopax_device` device)
- `cl_context get_cl_context` (const `goopax_device` device)
- `cl_command_queue get_cl_queue` (const `goopax_device` device)
- `cl_device_id get_cl_device` (const `goopax_device` device)
- `vector< string > get_cl_platform_extensions` (const `goopax_device` device)
- `vector< string > get_cl_device_extensions` (const `goopax_device` device)
- `bool have_cl_platform_extension` (const string &s, const `goopax_device` device)
- `bool have_cl_device_extension` (const string &s, const `goopax_device` device)
- `template<class BUF >`
`cl_mem get_cl_mem` (const BUF &buf)
- `goopax_device get_device_from_cl_queue` (`cl_command_queue` queue)
- static `image_buffer create_from_cl` (`goopax_device` device, `_cl_mem` *mem)
- static `image_array_buffer create_from_cl` (`goopax_device` device, `_cl_mem` *mem)

4.3.1 Detailed Description

OpenCL interoperability functions. These can be used to use existing OpenCL kernels in conjunction with GOOPAX.

4.3.2 Function Documentation

4.3.2.1 `create_from_cl()` [1/3]

```
static buffer create_from_cl (
    goopax_device device,
    _cl_mem * mem ) [inline], [static]
```

create buffer from OpenCL memory buffer

Parameters

<i>device</i>	GOOPAX device
<i>mem</i>	OpenCL memory buffer

4.3.2.2 `create_from_cl()` [2/3]

```
image_buffer< DIM, T, normalized > create_from_cl (
    goopax_device device,
    _cl_mem * mem ) [inline], [static]
```


Create image buffer from existing OpenCL image.

Parameters

<i>device</i>	goopax device that corresponds to the OpenCL device
<i>mem</i>	OpenCL memory resource

4.3.2.3 `create_from_cl()` [3/3]

```
image_array_buffer< DIM, T, normalized > create_from_cl (
    goopax_device device,
    _cl_mem * mem ) [inline], [static]
```

Create image array buffer from existing OpenCL image array.

Parameters

<i>device</i>	goopax device that corresponds to the OpenCL device
<i>mem</i>	OpenCL memory resource

4.3.2.4 `get_cl_context()`

```
cl_context goopax::impl::get_cl_context (
    const goopax_device device ) [inline]
```

Get OpenCL context from goopax device

4.3.2.5 `get_cl_device()`

```
cl_device_id goopax::impl::get_cl_device (
    const goopax_device device ) [inline]
```

Get OpenCL device from goopax device

4.3.2.6 `get_cl_device_extensions()`

```
vector<string> goopax::impl::get_cl_device_extensions (
    const goopax_device device ) [inline]
```

Get OpenCL device extensions for the given goopax device

4.3.2.7 `get_cl_mem()`

```
cl_mem goopax::impl::get_cl_mem (
    const BUF & buf ) [inline]
```

Returns OpenCL memory handle for the goopax buffer or [image_buffer](#)

Parameters

<i>buf</i>	buffer, image_buffer , or image_array_buffer
------------	--

4.3.2.8 `get_cl_platform()`

```
cl_platform_id goopax::impl::get_cl_platform (
    goopax_device device ) [inline]
```

Get OpenCL platform from goopax device

4.3.2.9 `get_cl_platform_extensions()`

```
vector<string> goopax::impl::get_cl_platform_extensions (
    const goopax_device device ) [inline]
```

Get OpenCL platform extensions for the given goopax device

4.3.2.10 `get_cl_queue()`

```
cl_command_queue goopax::impl::get_cl_queue (
    const goopax_device device ) [inline]
```

Get OpenCL queue from goopax device

4.3.2.11 `get_device_from_cl_queue()`

```
goopax_device goopax::impl::get_device_from_cl_queue (
    cl_command_queue queue ) [inline]
```

Create goopax device from OpenCL queue

4.3.2.12 `have_cl_device_extension()`

```
bool goopax::impl::have_cl_device_extension (
    const string & s,
    const goopax_device device ) [inline]
```

Test if OpenCL device extension is available for the device.

4.3.2.13 `have_cl_platform_extension()`

```
bool goopax::impl::have_cl_platform_extension (
    const string & s,
    const goopax_device device ) [inline]
```

Test if OpenCL platform extension is available for the device.

4.4 cpu and gpu functions

CPU functions In the namespace `goopax::extra_operators` additional function definitions for CPU data types are provided that otherwise are only provided for gpu data types.

Functions

- `template<typename T, typename = typename enable_if<is_integral<T>::value>::type>`
Tuint `popcount` (T i)
- `template<typename I, typename = typename enable_if<is_integral<I>::value>::type>`
I `rotl` (I a, Tint b)
- `template<typename I, typename = typename enable_if<is_integral<I>::value>::type>`
I `rotr` (I a, Tint b)
- double `sinpi` (double x)
- float `sinpi` (float x)
- half `sinpi` (half x)
- double `cospi` (double x)
- float `cospi` (float x)
- half `cospi` (half x)
- double `tanpi` (double x)
- float `tanpi` (float x)
- half `tanpi` (half x)
- double `asinpi` (double x)
- float `asinpi` (float x)
- half `asinpi` (half x)
- double `acospi` (double x)
- float `acospi` (float x)
- half `acospi` (half x)
- double `atanpi` (double x)
- float `atanpi` (float x)
- half `atanpi` (half x)
- `template<typename T, typename Enable >`
`gpu_type< T >` `min` (`gpu_type< T >` a, `gpu_type< T >` b)
- `template<typename T, typename Enable >`
`gpu_type< T >` `max` (`gpu_type< T >` a, `gpu_type< T >` b)
- `template<typename T >`
`gpu_type< T >` `clamp` (`gpu_type< T >` a, `typename enable_if< true, gpu_type< T >>::type` b, `typename enable_if< true, gpu_type< T >>::type` c)
- `template<typename T >`
`gpu_type< T >` `clamp` (`gpu_type< T >` a, `gpu_type< T >` b, `gpu_type< T >` c)
- `template<typename T >`
`gpu_type< T >` `min` (`gpu_type< T >` a, `typename unrangetype_core< T >::type` b)
- `template<typename T >`
`gpu_type< T >` `min` (`typename unrangetype_core< T >::type` a, `gpu_type< T >` b)
- `template<typename T >`
`gpu_type< T >` `max` (`gpu_type< T >` a, `typename unrangetype_core< T >::type` b)
- `template<typename T >`
`gpu_type< T >` `max` (`typename unrangetype_core< T >::type` a, `gpu_type< T >` b)
- `template<typename T, typename = typename enable_if<(is_integral<T>::value && !is_same<T, bool>::value)>::type>`
`gpu_type< typename make_unsigned< T >::type >` `abs` (`gpu_type< T >` a)
- `template<typename T, typename Enable = typename enable_if<(is_integral<T>::value && !is_same<T, bool>::value)>::type>`
`gpu_uint` `clz` (`gpu_type< T >` a)
- `template<typename T, typename Enable = typename enable_if<(is_integral<T>::value && !is_same<T, bool>::value)>::type>`
`gpu_uint` `popcount` (`gpu_type< T >` a)

- `template<typename T, typename Enable = typename enable_if<is_integral<T>::value && !is_same<T, bool>::value>::type> gpu_type< T > rotl (gpu_type< T > a, gpu_int b)`
- `template<typename T, typename Enable = typename enable_if<is_integral<T>::value && !is_same<T, bool>::value>::type> gpu_type< T > rotr (gpu_type< T > a, gpu_int b)`
- `template<typename T > Tuint clz (T i)`
- `Tuint clz (uint64_t i)`

4.4.1 Detailed Description

CPU functions In the namespace `goopax::extra_operators` additional function definitions for CPU data types are provided that otherwise are only provided for gpu data types.

4.4.2 Function Documentation

4.4.2.1 abs()

```
gpu_type<typename make_unsigned<T>::type> goopax::impl::abs (
    gpu_type< T > a )
```

returns absolute value. if `a` is an integer, the return type is unsigned.

4.4.2.2 acospi() [1/3]

```
double goopax::impl::acospi (
    double x ) [inline]
```

`acos(x) / PI`

4.4.2.3 acospi() [2/3]

```
float goopax::impl::acospi (
    float x ) [inline]
```

`acos(x) / PI`

4.4.2.4 acospi() [3/3]

```
half goopax::impl::acospi (
    half x ) [inline]
```

`acos(x) / PI`

4.4.2.5 asinpi() [1/3]

```
double goopax::impl::asinpi (
    double x ) [inline]
```

asin(x) / PI

4.4.2.6 asinpi() [2/3]

```
float goopax::impl::asinpi (
    float x ) [inline]
```

asin(x) / PI

4.4.2.7 asinpi() [3/3]

```
half goopax::impl::asinpi (
    half x ) [inline]
```

asin(x) / PI

4.4.2.8 atanpi() [1/3]

```
double goopax::impl::atanpi (
    double x ) [inline]
```

acos(x) / PI

4.4.2.9 atanpi() [2/3]

```
float goopax::impl::atanpi (
    float x ) [inline]
```

acos(x) / PI

4.4.2.10 atanpi() [3/3]

```
half goopax::impl::atanpi (
    half x ) [inline]
```

acos(x) / PI

4.4.2.11 clamp() [1/2]

```
gpu_type<T> goopax::impl::clamp (
    gpu_type< T > a,
    typename enable_if< true, gpu_type< T >>::type b,
    typename enable_if< true, gpu_type< T >>::type c )
```

clamping a to range (b,c). Similar to max(min(a, c), b)

4.4.2.12 clamp() [2/2]

```
gpu_type<T> goopax::impl::clamp (
    gpu_type< T > a,
    gpu_type< T > b,
    gpu_type< T > c )
```

clamping a to range (b,c). Similar to max(min(a, c), b)

4.4.2.13 clz() [1/3]

```
Tuint goopax::impl::clz (
    T i )
```

Counts the number of leading zeros in the given value of integral type

4.4.2.14 clz() [2/3]

```
Tuint goopax::impl::clz (
    uint64_t i ) [inline]
```

Counts the number of leading zeros in the given value of integral type

4.4.2.15 clz() [3/3]

```
gpu_uint goopax::impl::clz (
    gpu_type< T > a )
```

Counts the number of leading zeros in the given value of integral type

4.4.2.16 cospi() [1/3]

```
double goopax::impl::cospi (
    double x ) [inline]
```

cos(x*PI)

4.4.2.17 cospi() [2/3]

```
float goopax::impl::cospi (
    float x ) [inline]
```

cos(x*PI)

4.4.2.18 cospi() [3/3]

```
half goopax::impl::cospi (
    half x ) [inline]
```

cos(x*PI)

4.4.2.19 max() [1/3]

```
gpu_type< T > max (
    gpu_type< T > a,
    gpu_type< T > b )
```

maximum of both numbers

4.4.2.20 max() [2/3]

```
gpu_type<T> goopax::impl::max (
    gpu_type< T > a,
    typename unrangetype_core< T >::type b )
```

4.4.2.21 max() [3/3]

```
gpu_type<T> goopax::impl::max (
    typename unrangetype_core< T >::type a,
    gpu_type< T > b )
```

4.4.2.22 min() [1/3]

```
gpu_type< T > min (
    gpu_type< T > a,
    gpu_type< T > b )
```

minimum of both numbers

4.4.2.23 min() [2/3]

```
gpu_type<T> goopax::impl::min (
    gpu_type< T > a,
    typename unrangetype_core< T >::type b )
```

4.4.2.24 min() [3/3]

```
gpu_type<T> goopax::impl::min (
    typename unrangetype_core< T >::type a,
    gpu_type< T > b )
```


4.4.2.25 popcount() [1/2]

```
Tuint goopax::impl::popcount (  
    T i )
```

Counts the number of bits in the given value of integral type

4.4.2.26 popcount() [2/2]

```
gpu_uint goopax::impl::popcount (  
    gpu_type< T > a )
```

Counts the number of bits

4.4.2.27 rotl() [1/2]

```
I goopax::impl::rotl (  
    I a,  
    Tint b )
```

rotate left

4.4.2.28 rotl() [2/2]

```
gpu_type<T> goopax::impl::rotl (  
    gpu_type< T > a,  
    gpu_int b )
```

rotate left

4.4.2.29 rotr() [1/2]

```
I goopax::impl::rotr (  
    I a,  
    Tint b )
```

rotate right

4.4.2.30 rotr() [2/2]

```
gpu_type<T> goopax::impl::rotr (  
    gpu_type< T > a,  
    gpu_int b )
```

rotate right

4.4.2.31 sinpi() [1/3]

```
double goopax::impl::sinpi (  
    double x ) [inline]
```

sin(x*PI)

4.4.2.32 sinpi() [2/3]

```
float goopax::impl::sinpi (  
    float x ) [inline]
```

sin(x*PI)

4.4.2.33 sinpi() [3/3]

```
half goopax::impl::sinpi (  
    half x ) [inline]
```

sin(x*PI)

4.4.2.34 tanpi() [1/3]

```
double goopax::impl::tanpi (  
    double x ) [inline]
```

tan(x*PI)

4.4.2.35 tanpi() [2/3]

```
float goopax::impl::tanpi (  
    float x ) [inline]
```

tan(x*PI)

4.4.2.36 tanpi() [3/3]

```
half goopax::impl::tanpi (  
    half x ) [inline]
```

tan(x*PI)

4.5 CUDA interoperability

Functions

- static `buffer create_from_cuda` (`goopax_device` device, `void *mem`, `size_t size`)
- CUdevice `get_cuda_device` (`goopax_device` device)

4.5.1 Detailed Description

4.5.2 Function Documentation

4.5.2.1 `create_from_cuda()`

```
static buffer create_from_cuda (
    goopax_device device,
    void * mem,
    size_t size ) [inline], [static]
```

create buffer from CUDA buffer

Parameters

<i>device</i>	GOOPAX device
<i>pointer</i>	to CUDA device memory
<i>size</i>	size in number of elements of type T

4.5.2.2 `get_cuda_device()`

```
CUdevice goopax::impl::get_cuda_device (
    goopax_device device ) [inline]
```

get CUDA device from GOOPAX device

Parameters

<i>device</i>	GOOPAX device
---------------	---------------

Returns

CUDA device

4.6 Thread numbers

querying the current thread id

- [gpu_uint thread_id_in_warp \(\)](#)
- [gpu_uint warp_id_in_group \(\)](#)
- [gpu_uint warp_id \(\)](#)
- [gpu_uint local_id \(\)](#)
- [gpu_uint group_id \(\)](#)
- [gpu_uint global_id \(\)](#)

querying the thread layout of the current kernel

- [Tuint warp_size \(\)](#)
- [Tuint local_size \(\)](#)
- [Tuint num_groups \(\)](#)
- [Tuint global_size \(\)](#)
- [Tuint num_subthreads \(\)](#)
- [Tsize_t max_local_mem \(\)](#)

4.6.1 Detailed Description

These functions can be used to query the thread layout, the number of threads, and the id of the current thread. All functions may only be called within a kernel.

The threads are organized hierarchically:

- Programming normally occurs at the level of individual threads.
- A warp contains individual threads that execute SIMD instructions. Threads within a warp cannot branch into if clauses or loops without all other threads in the warp waiting. The functions [ballot\(\)](#) and [shuffle\(\)](#) can be used for fast inner-warp communication
- A workgroup contains one or more warps. Threads within a workgroup can communicate via local memory
- The total number of threads consists of multiple workgroups. Threads of different workgroups can only communicate via global memory by using atomic access functions.

4.6.2 Function Documentation

4.6.2.1 [global_id\(\)](#)

```
gpu_uint goopax::impl::gpu_thread::global_id ( ) [inline]
```

Returns the total thread ID on the GPU. Equals [group_id\(\)*local_size\(\) + local_id\(\)](#). May only be called in a GPU kernel.

4.6.2.2 global_size()

```
Tuint goopax::impl::gpu_thread::global_size ( ) [inline]
```

Returns the total number of threads on the GPU. Equals `local_size()*num_groups()`. May only be called in a GPU kernel

4.6.2.3 group_id()

```
gpu_uint goopax::impl::gpu_thread::group_id ( ) [inline]
```

Returns the workgroup ID. May only be called in a GPU kernel.

4.6.2.4 local_id()

```
gpu_uint local_id ( ) [inline]
```

Returns the local thread ID in the workgroup. May only be called in a GPU kernel.

Returns the local thread ID in the workgroup. May only be called in a GPU kernel.

4.6.2.5 local_size()

```
Tuint local_size ( ) [inline]
```

Returns the workgroup size. May only be called in a GPU kernel

Returns the workgroup size. May only be called in a GPU kernel

4.6.2.6 max_local_mem()

```
Tsize_t goopax::impl::gpu_thread::max_local_mem ( ) [inline]
```

Returns the maximum amount of local memory for each workgroup. May only be called in a GPU kernel

4.6.2.7 num_groups()

```
Tuint goopax::impl::gpu_thread::num_groups ( ) [inline]
```

Returns the number of workgroups. May only be called in a GPU kernel

4.6.2.8 num_subthreads()

```
Tuint goopax::impl::gpu_thread::num_subthreads ( ) [inline]
```

Returns the instruction level parallelization that works best for the GPU. May only be called in a GPU kernel

4.6.2.9 thread_id_in_warp()

```
gpu_uint goopax::impl::gpu_thread::thread_id_in_warp ( ) [inline]
```

Returns the id in the current warp. May only be called in a GPU kernel.

4.6.2.10 warp_id()

```
gpu_uint goopax::impl::gpu_thread::warp_id ( ) [inline]
```

Returns the id in the current warp. May only be called in a GPU kernel.

4.6.2.11 warp_id_in_group()

```
gpu_uint goopax::impl::gpu_thread::warp_id_in_group ( ) [inline]
```

Returns the warp id in the current group. May only be called in a GPU kernel.

4.6.2.12 warp_size()

```
Tuint warp_size ( ) [inline]
```

Returns the warp size of the device for which the kernel is being built. May only be called in a GPU kernel. The warp size is typically the SIMD instruction width. Threads within a warp cannot branch into loops and if-clauses without all other threads waiting for them to finish.

Returns the warp size of the device for which the kernel is being built. May only be called in a GPU kernel. The warp size is typically the SIMD instruction width. Threads within a warp cannot branch into loops and if-clauses without all other threads waiting for them to finish.

4.7 Device

Classes

- class `goopax_device`
Goopax device. This class is used to reference the physical devices in the system.

Enumerations

- enum `envmode` { ,
`env_CPU = 1, env_CL = 2, env_CUDA = 4, env_METAL = 8,`
`env_GPU = 0xfffe, env_ALL = 0xffffffff }`

Functions

- `envmode operator| (envmode a, envmode b)`
- `goopax_device get_current_build_device ()`
- `vector< goopax_device > devices (envmode backend=env_GPU)`
- `goopax_device default_device (envmode backend=env_GPU)`
- `void init_params (int &argc, char **&argv, const char *paramfile=NULLptr)`

4.7.1 Detailed Description

4.7.2 Enumeration Type Documentation

4.7.2.1 envmode

enum `envmode`

Specifies which backend to use. When multiple backends are set, CUDA and METAL will be preferred over CL for devices that support them.

Enumerator

<code>env_CPU</code>	CPU backend.
<code>env_CL</code>	OpenCL backend.
<code>env_CUDA</code>	CUDA backend.
<code>env_METAL</code>	Metal backend.
<code>env_GPU</code>	Any GPU backend.
<code>env_ALL</code>	Any backend, CPU and GPU.

4.7.3 Function Documentation

4.7.3.1 default_device()

```
goopax_device goopax::impl::default_device (
    envmode backend = env_GPU ) [inline]
```

Get the fastest available device for the given backend(s)

Returns

best suitable device. If no usable device is found, the returned value will be default-constructed, and `goopax_device::valid()` will return false on it.

Parameters

<i>backend</i>	which backend(s) to use.
----------------	--------------------------

4.7.3.2 devices()

```
vector<goopax_device> goopax::impl::devices (
    envmode backend = env_GPU ) [inline]
```

Get vector of available devices for the given backend(s) Devices that are both CUDA and OpenGL capable will show up both for the `env_CUDA` and for the `env_CL` backend, but they will only be returned once if the both `env_CL` and `env_CUDA` flags are selected. Equally, devices that support the Metal backend may also show up if the `env_CL` backend is selected without the `env_METAL` backend.

Returns

vector of devices. If no usable devices are found, an empty list is returned.

Parameters

<i>backend</i>	which backend(s) to use.
----------------	--------------------------

4.7.3.3 get_current_build_device()

```
goopax_device goopax::impl::get_current_build_device ( ) [inline]
```

Returns the device for which the current kernel is being built. Only valid while building kernel.

4.7.3.4 init_params()

```
void goopax::impl::init_params (
    int & argc,
```



```
char **& argv,  
const char * paramfile = nullptr ) [inline]
```

May be called to allow passing command line arguments to GOOPAX or to PARAM or PARAMOPT user values.

Parameters

<i>argc</i>	argc from the main() function
<i>argv</i>	argv from the main() function
<i>paramfile</i>	optional filename of configuration file

4.7.3.5 operator" | ()

```
envmode goopax::operator| (  
    envmode a,  
    envmode b ) [inline]
```

4.8 Gather return values

Classes

- struct [gather_add< T >](#)
- struct [gather_min< T >](#)
- struct [gather_max< T >](#)

4.8.1 Detailed Description

gather types. These types can be used to gather values from the GPU threads to return values from GPU kernels. Some types are predefined here. User-defined types can be added accordingly. See the predefined types in `gather.h` for how this is done. If used as return type or reference argument of the `program()` function of GPU kernels, values are returned as [goopax_future](#) when the kernel is called.

4.9 OpenGL interoperability

Functions

- void `flush_gl_interop` (`goopax_device` device)
- static `buffer create_from_gl` (`goopax_device` device, unsigned int `GLres`, `BUFFER_FLAGS` flags=`BUFFER_READ_WRITE`)
- static `image_buffer create_from_gl` (`goopax_device` device, unsigned int `GLres`, `IMAGE_FLAGS` flags=`IMAGE_READ_WRITE`, unsigned int `GLtarget`=(`DIM==1` ? `0x0DE0` :(`DIM==2` ? `0x0DE1` :`0x806F`)), int `miplevel`=0)
- static `image_array_buffer create_from_gl` (`goopax_device` device, unsigned int `GLres`, `IMAGE_FLAGS` flags=`IMAGE_READ_WRITE`, unsigned int `GLtarget`=(`DIM==1` ? `0x0DE0` :(`DIM==2` ? `0x0DE1` :`0x806F`)), int `miplevel`=0)

4.9.1 Detailed Description

4.9.2 Function Documentation

4.9.2.1 `create_from_gl()` [1/3]

```
buffer< T, SIZE_TYPE > create_from_gl (
    goopax_device device,
    unsigned int GLres,
    BUFFER_FLAGS flags = BUFFER_READ_WRITE ) [inline], [static]
```

Create goopax buffer from existing OpenGL buffer

Parameters

<i>device</i>	goopax device
<i>GLres</i>	buffer ID of OpenGL buffer
<i>flags</i>	access type

4.9.2.2 `create_from_gl()` [2/3]

```
image_buffer< DIM, T, normalized > create_from_gl (
    goopax_device device,
    unsigned int GLres,
    IMAGE_FLAGS flags = IMAGE_READ_WRITE,
    unsigned int GLtarget = (DIM == 1 ? 0x0DE0 : (DIM == 2 ? 0x0DE1 : 0x806F)),
    int miplevel = 0 ) [inline], [static]
```

Creates image that is linked to existing OpenGL image.

Parameters

<i>device</i>	goopax device that contains the OpenGL image
<i>GLres</i>	OpenGL resource ID
<i>flags</i>	access type. Must be one of IMAGE_READ_WRITE, IMAGE_READ_ONLY, IMAGE_WRITE_ONLY, IMAGE_WRITE_DISCARD
<i>GLtarget</i>	OpenGL target. Defaults to GL_TEXTURE_1D or GL_TEXTURE_2D for 1D/2D images.
<i>miplevel</i>	OpenGL miplevel of the texture. Defaults to 0.

4.9.2.3 create_from_gl() [3/3]

```
image_array_buffer< DIM, T, normalized > create_from_gl (
    goopax_device device,
    unsigned int GLres,
    IMAGE_FLAGS flags = IMAGE_READ_WRITE,
    unsigned int GLtarget = (DIM == 1 ? 0x0DE0 : (DIM == 2 ? 0x0DE1 : 0x806F)),
    int miplevel = 0 ) [inline], [static]
```

Creates image array that is linked to existing OpenGL image.

Parameters

<i>device</i>	goopax device that contains the OpenGL image
<i>GLres</i>	OpenGL resource ID
<i>flags</i>	access type. Must be one of IMAGE_READ_WRITE, IMAGE_READ_ONLY, IMAGE_WRITE_ONLY, IMAGE_WRITE_DISCARD
<i>GLtarget</i>	OpenGL target. Defaults to GL_TEXTURE_1D or GL_TEXTURE_2D for 1D/2D images.
<i>miplevel</i>	OpenGL miplevel of the texture. Defaults to 0.

4.9.2.4 flush_gl_interop()

```
void goopax::impl::flush_gl_interop (
    goopax_device device ) [inline]
```

Must be called before calling OpenGL functions on shared OpenGL buffers or images that exist on the given device.

4.10 Math

Functions

- `template<typename T, typename = typename enable_if<is_floating_point<T>::value>::type> gpu_type< T > pow (gpu_type< T > a, int ENUM, unsigned int DENOM=1)`
- `template<int ENUM, unsigned int DENOM, class T, class ENABLE = typename enable_if<is_floating_point<T>::value>::type> gpu_type< T > pow (const gpu_type< T > a)`
- `template<class T, class ENABLE = typename enable_if<is_floating_point<T>::value>::type> gpu_type< T > pow (const gpu_type< T > a, const gpu_type< T > b)`
- `template<class T, class D, class ENABLE = typename enable_if<is_floating_point<T>::value && is_floating_point<D>::value>::type> gpu_type< T > pow (const gpu_type< T > a, const D b)`
- `gpu_half sqrt (gpu_half a)`
Square root.
- `gpu_float sqrt (gpu_float a)`
Square root.
- `gpu_double sqrt (gpu_double a)`
Square root.
- `gpu_half cbrt (gpu_half a)`
Cubic root.
- `gpu_float cbrt (gpu_float a)`
Cubic root.
- `gpu_double cbrt (gpu_double a)`
Cubic root.

4.10.1 Detailed Description

4.10.2 Function Documentation

4.10.2.1 pow() [1/4]

```
gpu_type<T> goopax::impl::pow (
    gpu_type< T > a,
    int ENUM,
    unsigned int DENOM = 1 ) [inline]
```

Returns a to the power (ENUM/DENOM).

4.10.2.2 pow() [2/4]

```
gpu_type<T> goopax::impl::pow (
    const gpu_type< T > a )
```

Returns a to the power (ENUM/DENOM).

4.10.2.3 pow() [3/4]

```
gpu_type<T> goopax::impl::pow (
    const gpu_type< T > a,
    const gpu_type< T > b )
```

Returns a to the power b.

4.10.2.4 pow() [4/4]

```
gpu_type<T> goopax::impl::pow (
    const gpu_type< T > a,
    const D b )
```

Returns a to the power b.

4.11 debugging

Classes

- class [gpu_ostream](#)

Macros

- #define [gpu_assert](#)(...) ::goopax::impl::gpu_assert_impl((__VA_ARGS__), __FILE__, __LINE__, #__VA_ARGS__);

4.11.1 Detailed Description

For debugging, various methods are provided. The `gpu_ostream` streaming class can be used to provide output from within a kernel. The `gpu_assert` macro can be placed in kernels for validity checks. The `checknan` data type does extensive automatic error checks on memory access.

4.11.2 Macro Definition Documentation

4.11.2.1 `gpu_assert`

```
#define gpu_assert(  
    ... ) ::goopax::impl::gpu_assert_impl((__VA_ARGS__), __FILE__, __LINE__, #__VA_ARGS__);
```

asserts that the given value of type `gpu_bool` is true. If not, an error message is displayed, and an exception is thrown.

4.12 Types

Classes

- struct [gettype< T, EIF >](#)

type info and type conversion mechanism With the `gettype` struct, CPU and GPU types can be converted to each other, and type information obtained `typename gettype<T>::cpu` converts `T` to the corresponding CPU type. `T` can be CPU type, GPU type, or CPU debug type, or a `goopax` struct type. `typename gettype<T>::gpu` converts `T` to the corresponding GPU type. `T` can be CPU type, GPU type, or CPU debug type, or a `goopax` struct type. `gettype<↔T>::size` provides the size of type `T` in bytes. `T` can be CPU type, GPU type, or CPU debug type, or a `goopax` struct type. `typename gettype<T>::template change<U>::type` changes the type of `T` to type `U` while preserving CPU/↔GPU/debug mode. If `T` is CPU type, the resulting type is also a CPU type. If `T` is GPU type, the resulting type is a GPU type. If `T` is debug data type, the resulting type is also a debug data type.

- class [gpu_type< T, scope >](#)

4.12.1 Detailed Description

gpu types, used in gpu kernels.


```

    const A & a,
    const A & b,
    typename enable_if< is_gpu_struct< A >::value &&is_checknan< typename get_X_↵
type< A >::type >::value >::type * = nullptr )

```

conditional operator.

Parameters

<i>c</i>	condition
<i>a</i>	input value a
<i>b</i>	input value b
<i>return</i>	<i>c ? a : b</i>

4.13.2.3 cond() [3/5]

```

gpu_type< T > cond (
    const gpu_bool & c,
    const gpu_type< T > & a,
    const gpu_type< T > & b ) [inline]

```

conditional operator conditional function. Returns *c ? a : b*

conditional function. Returns *c ? a : b*

4.13.2.4 cond() [4/5]

```

A goopax::impl::cond (
    bool c,
    const A & a,
    const A & b,
    typename enable_if< is_gpu_struct< A >::value &&is_same< typename get_X_type< A
>::type, xtype_int >::value >::type * = nullptr )

```

conditional operator.

Parameters

<i>c</i>	condition
<i>a</i>	input value a
<i>b</i>	input value b
<i>return</i>	<i>c ? a : b</i>

4.13.2.5 cond() [5/5]

```

auto goopax::impl::cond (
    const BOOL & c,

```

```
    const V1 & a,  
    const V2 & b ) -> typename changetype<V1, typename remove_reference<decltype(cond(c,  
a[0], b[0]))>::type>::type    [inline]
```

conditional function, overload for vectors and arrays

4.14 Image

Classes

- class [image_buffer_map](#)< DIM, T, is_const >
- class [image_array_buffer_map](#)< DIM, T, is_const >
- class [image_buffer](#)< DIM, T, normalized >
- class [image_array_buffer](#)< DIM, T, normalized >
- class [image_resource](#)< DIM, T, normalized >
- class [image_array_resource](#)< DIM, T, normalized >

Typedefs

- `template<unsigned int DIM, typename T >`
using [const_image_buffer_map](#) = [image_buffer_map](#)< DIM, T, true >

Enumerations

- enum [IMAGE_FLAGS](#) { [IMAGE_READ_WRITE](#) = 1, [IMAGE_READ_ONLY](#) = 2, [IMAGE_WRITE_ONLY](#) = 4, [IMAGE_WRITE_DISCARD](#) = 8 }

4.14.1 Detailed Description

4.14.2 Enumeration Type Documentation

4.14.2.1 IMAGE_FLAGS

enum [IMAGE_FLAGS](#)

Image access flags

Enumerator

IMAGE_READ_WRITE	read/write access
IMAGE_READ_ONLY	read-only access
IMAGE_WRITE_ONLY	write-only access
IMAGE_WRITE_DISCARD	write-only, previous values are not preserved

4.15 Kernel

Classes

- class [goopax_future_info](#)
- class [goopax_future< T >](#)
- class [goopax_future< void >](#)
- struct [full_kernel](#)
- struct [kernel< K >](#)

kernel class to derive gpu kernels from

4.15.1 Detailed Description

GPU kernels are inherited from kernel. The entry point to the GPU kernel is provided by the function 'program'. However, GPU kernels can call other functions, so they are not limited to the body of function 'program'.

Example:

```
struct my_kernel :
    kernel<my_kernel>
{
    void program(resource<int>& A)
    {
        A[global_id()] = 5;    // This is executed on the GPU
    }
};
```

4.16 Loops

Macros

- `#define gpu_if(...)` for (goopax::impl::gpu_if_clause __goopax_if_obj(__FILE__, __LINE__, static ↵ cast<goopax::gpu_bool>(__VA_ARGS__)); __goopax_if_obj.usecount == 0; ++__goopax_if_obj.usecount)
- `#define gpu_elseif(...)` for (goopax::impl::gpu_if_clause __goopax_if_obj(__FILE__, __LINE__, 1, static ↵ cast<goopax::gpu_bool>(__VA_ARGS__)); __goopax_if_obj.usecount == 0; ++__goopax_if_obj.usecount)
- `#define gpu_else gpu_elseif(true)`
- `#define gpu_while(...)` for (goopax::impl::infinite_loop LOOP(__FILE__, __LINE__, __VA_ARGS__); LOO ↵ P.loop_do_continue(); LOOP.loop_set_next(__VA_ARGS__))
- `#define gpu_for(...)` goopax_gpu_for(0, __VA_ARGS__)
gpu for loop.
usage: gpu_for(<GPU TYPE> <var>=<start>, <end> [,<step>]) { ... }
- `#define gpu_for_local(...)` goopax_gpu_for(1, __VA_ARGS__)
- `#define gpu_for_global(...)` goopax_gpu_for(2, __VA_ARGS__)
- `#define gpu_for_group(...)` goopax_gpu_for(3, __VA_ARGS__)

4.16.1 Detailed Description

Loops in gpu kernels can be achieved by the use of `gpu_for` or `gpu_while` statements. Alternatively, small loops that don't depend on gpu variables may also be implemented as standard C++ `for/while` loops. This will lead to explicit unrolling of the loop.

The usage of the `gpu_for` loop is different from C++ `for` loops and looks like this:

```
gpu_for(gpu_uint k=0, 10)
{
  <do something> // k runs from 0 to 9
}
```

Step sizes can be provided as a third argument:

```
gpu_for(gpu_int k=0, 10, 2)
{
  <do something> // only runs over even numbers form 0 to 8
}
```

The comparison operator can be specified as a fourth argument as a C string (defaults to "<="):

```
gpu_for(gpu_int k=0, 10, 2, "<=")
{
  <do something> // only runs over even numbers form 0 to 10
}
```

4.16.2 Macro Definition Documentation

4.16.2.1 gpu_else

```
#define gpu_else gpu_elseif(true)
```

`gpu_elseif` must follow a `gpu_if` or `gpu_elseif` block. The block following the `gpu_elseif` statement is executed by all threads that did not execute the previous if blocks.

4.16.2.2 gpu_elseif

```
#define gpu_elseif(
    ... ) for (goopax::impl::gpu_if_clause __goopax_if_obj(__FILE__, __LINE__, 1,
static_cast<goopax::gpu_bool>(__VA_ARGS__)); __goopax_if_obj.usecount == 0; ++__goopax_if_↔
obj.usecount)
```

`gpu_elseif`, must follow a `gpu_if` or `gpu_elseif` block. The block following the `gpu_elseif()` statement is executed by all threads for which the argument is true.

4.16.2.3 gpu_for_global

```
#define gpu_for_global(
    ... ) goopax_gpu_for(2, __VA_ARGS__)
```

gpu loop, parallelized over all threads

4.16.2.4 gpu_for_group

```
#define gpu_for_group(
    ... ) goopax_gpu_for(3, __VA_ARGS__)
```

gpu loop, parallelized over workgroups

4.16.2.5 gpu_for_local

```
#define gpu_for_local(
    ... ) goopax_gpu_for(1, __VA_ARGS__)
```

gpu loop, parallelized over threads in workgroup.

4.16.2.6 gpu_if

```
#define gpu_if(
    ... ) for (goopax::impl::gpu_if_clause __goopax_if_obj(__FILE__, __LINE__↔
, static_cast<goopax::gpu_bool>(__VA_ARGS__)); __goopax_if_obj.usecount == 0; ++__goopax_if_↔
obj.usecount)
```

`gpu_if`, to be used in kernels. Takes `gpu_bool` as input. The block following the `gpu_if()` statement is executed by all threads for which the argument is true.

4.16.2.7 gpu_while

```
#define gpu_while(
    ... ) for (goopax::impl::infinite_loop LOOP(__FILE__, __LINE__, __VA_ARGS__);
LOOP.loop_do_continue(); LOOP.loop_set_next(__VA_ARGS__))
```

while loop.

4.17 operators

4.17.1 Detailed Description

These functions are defined to add CPU pendants for GPU functions, for which no CPU function is provided by standard C++.

4.18 Reinterpret

Functions

- `template<class TO , class FROM >`
`TO reinterpret (const FROM &from)`
bit-wise conversion
Reinterpret bits into another type.
- `template<typename TO , typename TFROM , typename SIZE_FROM >`
`TO reinterpret (const buffer< TFROM, SIZE_FROM > &from)`
Reinterpreting buffer value type
Changes the buffer value type without changing the data content. The returned buffer points to the same memory buffer as the original buffer.

4.18.1 Detailed Description

The reinterpret functions provide functionalities for bit-wise conversion between data types without changing the underlying representation. Various types can be converted into each other: intrinsic types to goopax structs, goopax structs to `std::vector` of other goopax structs, vectors to intrinsic types, etc. The sizes of the source value and the destination value must match, otherwise the compiler will report an error. If the source type is an `std::vector`, the size is checked at runtime. If the destination type is an `std::vector`, its size is chosen so that it matches the source type. If the source type is an CPU type, the destination type must also be a CPU type. If the source type is a GPU type, the destination type must also be a GPU type.

4.18.2 Function Documentation

4.18.2.1 reinterpret() [1/2]

```
TO reinterpret (
    const FROM & from ) [inline]
```

bit-wise conversion

Reinterpret bits into another type.

Parameters

<i>TO</i>	destination type
<i>from</i>	source This function reinterprets the input parameter into another type. FROM and TO must have the same size. They may be CPU type, GPU type, <code>std::vector</code> , or goopax struct type.

Example: `gpu_double a = 17; std::array<gpu_int, 2> b = reinterpret<std::array<gpu_int, 2> >(a); gpu_uint64 c = reinterpret<gpu_uint64> c;`

4.18.2.2 reinterpret() [2/2]

```
TO reinterpret (
```

```
const buffer< TFROM, SIZE_FROM > & from ) [inline]
```

Reinterpreting buffer value type

Changes the buffer value type without changing the data content. The returned buffer points to the same memory buffer as the original buffer.

Parameters

<i>TO</i>	destination type
<i>from</i>	source buffer

4.19 random numbers

Classes

- class [WELL512](#)
- class [WELL512lib](#)

4.19.1 Detailed Description

Random numbers can be generated with the WELL512 random number generator. The current implementation is optimized for high throughput, but it has the limitation that random numbers are generated in 512 bit blocks. To use the random number generator, a pair of classes need to be instantiated: The WELL512 class in host code and the WELL512lib class in device code. The WELL512 class is used for storing the random seed between kernel calls, the WELL512lib class provides the actual generator of the random numbers. See the pi example program for an example implementation.

4.20 goopax struct types

Classes

- struct `goopax_struct_type< std::array< T, N > >`
std::array and derived classes

Functions

- `template<typename T >`
`output_goopax_struct_helper< T >` `output_goopax_struct` (const T &t)

4.20.1 Detailed Description

here some C++ types are prepared to be used as goopax struct. Other struct types need to be prepared in a similar way before they can be used to store memory in goopax.

4.20.2 Function Documentation

4.20.2.1 `output_goopax_struct()`

```
output_goopax_struct_helper< T > output_goopax_struct (
    const T & t )
```

Outputs the contents of a goopax struct.

Outputs the contents of a goopax struct.

4.21 Thread_communication

Functions

- `template<typename T, typename = typename enable_if<is_same<T, typename gettype<T>::gpu>::value>::type> T shuffle (T value, gpu_uint source_thread, Tuint width)`
Exchanges values between threads in the workgroup.
- `vector< gpu_uint > ballot (gpu_bool value, Tuint width)`
collect booleans from threads of the workgroup into integer bitset
- `gpu_bool work_group_all (gpu_bool b)`
- `gpu_bool work_group_any (gpu_bool b)`
- `template<typename G > G::gpu_value_type work_group_reduce (typename G::gpu_value_type value, Tuint width)`
- `template<typename T, typename = typename enable_if<is_same<T, bool>::value>::type> gpu_type< T > work_group_reduce_add (gpu_type< T > b, Tuint width)`
- `template<typename gpuT > gpuT work_group_reduce_add (gpuT b, Tuint width)`
- `template<typename T > gpu_type< T > work_group_reduce_min (gpu_type< T > b, Tuint width)`
- `template<typename gpuT > gpuT work_group_reduce_min (gpuT b, Tuint width)`
- `template<typename T > gpu_type< T > work_group_reduce_max (gpu_type< T > b, Tuint width)`
- `template<typename gpuT > gpuT work_group_reduce_max (gpuT b, Tuint width)`
- `template<typename T > gpu_type< T > work_group_broadcast (gpu_type< T > val, gpu_uint src_id)`
- `template<typename T > gpu_type< T > work_group_scan_exclusive_add (gpu_type< T > val)`
- `template<typename T > gpu_type< T > work_group_scan_inclusive_add (gpu_type< T > val)`
- `template<typename T > gpu_type< T > work_group_scan_exclusive_min (gpu_type< T > val)`
- `template<typename T > gpu_type< T > work_group_scan_inclusive_min (gpu_type< T > val)`
- `template<typename T > gpu_type< T > work_group_scan_exclusive_max (gpu_type< T > val)`
- `template<typename T > gpu_type< T > work_group_scan_inclusive_max (gpu_type< T > val)`

4.21.1 Detailed Description

These functions are provided to exchange data between threads in the workgroup.

4.21.2 Function Documentation

4.21.2.1 ballot()

```
vector<gpu\_uint> goopax::impl::ballot (
    gpu\_bool value,
    Tuint width ) [inline]
```

collect booleans from threads of the workgroup into integer bitset

Parameters

<i>value</i>	boolean to contribute from the current thread
<i>width</i>	sub-group width for the ballot operation. width must be a power of 2 and must not exceed <code>local_size()</code> . For nvidia GPUs using the CUDA backend, it is suggested that width has a value of 32 or less, in order to take advantage of CUDA's warp shuffle functions.

Returns

bitset of the threads involved. If width \leq 32, a vector of size 1 is returned, otherwise a vector of size width/32 is returned.

4.21.2.2 shuffle()

```
T goopax::impl::shuffle (
    T value,
    gpu_uint source_thread,
    Tuint width )
```

Exchanges values between threads in the workgroup.

value T goopax struct data type

Parameters

<i>value</i>	value to share to other threads
<i>source_thread</i>	the value is read from this thread within the sub-group. source_thread must have a value between 0 and width-1.
<i>width</i>	sub-group width in which to exchange the values. width must be a power of 2 and/or equal to <code>local_size()</code> . width must not exceed <code>local_size()</code> . For nvidia GPUs using the CUDA backend, it is suggested that width has a value of 32 or less, in order to take advantage of CUDA's warp shuffle functions.

Returns

value read from thread 'source_thread'

4.21.2.3 work_group_all()

```
gpu_bool goopax::impl::work_group_all (
    gpu_bool b ) [inline]
```

Returns true if b is true for all threads in the workgroup.

Parameters

<i>b</i>	input value
----------	-------------

Returns

the resulting boolean or over the threads

4.21.2.4 work_group_any()

```
gpu_bool goopax::impl::work_group_any (
    gpu_bool b ) [inline]
```

Returns true if *b* is true for any thread in the workgroup.

Parameters

<i>b</i>	input value
----------	-------------

Returns

the resulting boolean and over the threads

4.21.2.5 work_group_broadcast()

```
gpu_type<T> goopax::impl::work_group_broadcast (
    gpu_type< T > val,
    gpu_uint src_id )
```

broadcast value from one thread to all threads in the workgroup

Parameters

<i>val</i>	source value
<i>src</i> ↔ <i>_id</i>	thread number to read value from

Returns

value that thread '*src_id*' has provided

4.21.2.6 work_group_reduce()

```
G::gpu_value_type work_group_reduce (
    typename G::gpu_value_type value,
    Tuint width )
```

reduce operation for user defined gather operator

Parameters

<i>G</i>	gather operator type. See the pre-defined gather types in the file 'goopax/gather.h' as examples.
<i>value</i>	input value from the current thread. May be any gpu type, including goopax struct types
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and/or equal to local_size() . width must not exceed local_size() .

Returns

result of the gather operation

4.21.2.7 work_group_reduce_add() [1/2]

```
gpu_type<T> goopax::impl::work_group_reduce_add (
    gpu_type< T > b,
    Tuint width )
```

reduce operation 'add'

Parameters

<i>value</i>	input value
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and must not exceed local_size() .
<i>return</i>	result of the gather operation

4.21.2.8 work_group_reduce_add() [2/2]

```
gpuT goopax::impl::work_group_reduce_add (
    gpuT b,
    Tuint width )
```

reduce operation 'add'

Parameters

<i>value</i>	input value
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and must not exceed local_size() .
<i>return</i>	result of the gather operation

4.21.2.9 `work_group_reduce_max()` [1/2]

```
gpu_type<T> goopax::impl::work_group_reduce_max (
    gpu_type< T > b,
    Tuint width )
```

reduce operation 'max'

Parameters

<i>value</i>	input value
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and must not exceed local_size() .
<i>return</i>	result of the gather operation

4.21.2.10 `work_group_reduce_max()` [2/2]

```
gpuT goopax::impl::work_group_reduce_max (
    gpuT b,
    Tuint width )
```

reduce operation 'max'

Parameters

<i>value</i>	input value
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and must not exceed local_size() .
<i>return</i>	result of the gather operation

4.21.2.11 `work_group_reduce_min()` [1/2]

```
gpu_type<T> goopax::impl::work_group_reduce_min (
    gpu_type< T > b,
    Tuint width )
```

reduce operation 'min'

Parameters

<i>value</i>	input value
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and must not exceed local_size() .
<i>return</i>	result of the gather operation

4.21.2.12 `work_group_reduce_min()` [2/2]

```
gpuT goopax::impl::work_group_reduce_min (
    gpuT b,
    Tuint width )
```

reduce operation 'min'

Parameters

<i>value</i>	input value
<i>width</i>	sub-group width for the gather operation. width must be a power of 2 and must not exceed local_size() .
<i>return</i>	result of the gather operation

4.21.2.13 `work_group_scan_exclusive_add()`

```
gpu_type<T> goopax::impl::work_group_scan_exclusive_add (
    gpu_type< T > val )
```

Does an exclusive scan operation.

Parameters

<i>val</i>	input value from the current thread
------------	-------------------------------------

Returns

the resulting value equals the sum of all values provided by threads 0 to [local_id\(\)](#)-1

4.21.2.14 `work_group_scan_exclusive_max()`

```
gpu_type<T> goopax::impl::work_group_scan_exclusive_max (
    gpu_type< T > val )
```

Does an exclusive scan operation.

Parameters

<i>val</i>	input value from the current thread
------------	-------------------------------------

Returns

the resulting value equals the maximum of all values provided by threads 0 to [local_id\(\)-1](#)

4.21.2.15 work_group_scan_exclusive_min()

```
gpu_type<T> goopax::impl::work_group_scan_exclusive_min (  
    gpu_type< T > val )
```

Does an exclusive scan operation.

Parameters

<i>val</i>	input value from the current thread
------------	-------------------------------------

Returns

the resulting value equals the minimum of all values provided by threads 0 to [local_id\(\)-1](#)

4.21.2.16 work_group_scan_inclusive_add()

```
gpu_type<T> goopax::impl::work_group_scan_inclusive_add (  
    gpu_type< T > val )
```

Does an inclusive scan operation.

Parameters

<i>val</i>	input value from the current thread
------------	-------------------------------------

Returns

the resulting value equals the sum of all values provided by threads 0 to [local_id\(\)](#)

4.21.2.17 work_group_scan_inclusive_max()

```
gpu_type<T> goopax::impl::work_group_scan_inclusive_max (  
    gpu_type< T > val )
```

Does an inclusive scan operation.

Parameters

<i>val</i>	input value from the current thread
------------	-------------------------------------

Returns

the resulting value equals the maximum of all values provided by threads 0 to [local_id\(\)](#)

4.21.2.18 work_group_scan_inclusive_min()

```
gpu_type<T> goopax::impl::work_group_scan_inclusive_min (  
    gpu_type< T > val )
```

Does an inclusive scan operation.

Parameters

<i>val</i>	input value from the current thread
------------	-------------------------------------

Returns

the resulting value equals the minimum of all values provided by threads 0 to [local_id\(\)](#)

Chapter 5

Class Documentation

5.1 `buffer< T, SIZE_TYPE >` Class Template Reference

Public Types

- using `resource_type` = `resource< T, SIZE_TYPE >`
type of resource that matches to the buffer type

Public Member Functions

- `goopax_device get_device ()` const
- `size_type size ()` const
- `template<typename SRC_SIZE_TYPE >`
`void copy (const buffer< T, SRC_SIZE_TYPE > &src, typename enable_if< true, SRC_SIZE_TYPE >::type count, SIZE_TYPE start_dest, typename enable_if< true, SRC_SIZE_TYPE >::type start_src)`
- `template<typename SRC_SIZE_TYPE >`
`void copy (const buffer< T, SRC_SIZE_TYPE > &src)`
- `void copy_to_host (T *p, size_t beginpos, size_t endpos)` const
- `void copy_to_host (T *p)` const
- `void copy_from_host (const T *p, size_t beginpos, size_t endpos)`
- `void copy_from_host (const T *p)`
- `template<class V, class ENABLE = typename enable_if<is_container<V>::value>::type>`
`buffer & operator= (const V &v)`
- `void fill (T value, size_type begin, size_type end)`
- `void fill (T value)`
- `template<typename = void>`
`T min (const size_type begin, const size_type end)` const
- `template<typename = void>`
`T min ()` const
- `template<typename = void>`
`T max (const size_type begin, const size_type end)` const
- `template<typename = void>`
`T max ()` const
- `template<typename = void>`
`T sum (const size_type begin, const size_type end)` const
- `template<typename = void>`
`T sum ()` const

- `buffer & operator= (initializer_list< T > il)`
- `vector< value_type > to_vector () const`
- `template<class V , class ENABLE = typename enable_if<is_container<V>::value>::type>`
`buffer (goopax_device device, const V &v)`
- `template<class IT , typename = typename std::iterator_traits<IT>::value_type>`
`buffer (goopax_device device, const IT &a, const IT &b)`
- `buffer (goopax_device device, initializer_list< value_type > il)`
- `buffer (goopax_device device, size_t size, T *host_ptr=nullptr, BUFFER_FLAGS flags=BUFFER_READ_↔`
`WRITE)`
- `buffer (buffer &&)=default`
move semantics
- `buffer & operator= (buffer &&)=default`
move semantics
- `buffer ()=default`

Static Public Member Functions

- static `buffer create_from_gl (goopax_device device, unsigned int GLres, BUFFER_FLAGS flags=BUFFER_↔`
`_READ_WRITE)`
- static `buffer create_from_cl (goopax_device device, _cl_mem *mem)`
- static `buffer create_from_cuda (goopax_device device, void *mem, size_t size)`

Friends

- `template<typename TO , typename TFROM , typename SIZE_FROM >`
`TO reinterpret (const buffer< TFROM, SIZE_FROM > &from)`
Reinterpreting buffer value type
Changes the buffer value type without changing the data content. The returned buffer points to the same memory buffer as the original buffer.
- `ostream & operator<< (ostream &s, const buffer &b)`

5.1.1 Detailed Description

```
template<typename T, typename SIZE_TYPE>
class goopax::impl::buffer< T, SIZE_TYPE >
```

buffer class. Used to allocate memory on the device

Template Parameters

<i>T</i>	value type. This can either be a CPU intrinsic type, or a goopax struct type.
<i>SIZE_TYPE</i>	This can be a 32 bit or 64 bit integer type. 32 bit types give better performance. 64 bit types must be used for very large memory buffers. Goopax will throw an exception if buffers are allocated that are too large for the given size type. The default is 32 bit unsigned int.
<i>is_const</i>	only read access is allowed. This has the same effect as using a const reference of the resource as kernel parameter.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 `buffer()` [1/5]

```
buffer (
    goopax_device device,
    const V & v ) [inline]
```

initialize from `std::vector`

5.1.2.2 `buffer()` [2/5]

```
buffer (
    goopax_device device,
    const IT & a,
    const IT & b ) [inline]
```

initialize from begin/end pointer

5.1.2.3 `buffer()` [3/5]

```
buffer (
    goopax_device device,
    initializer_list< value_type > il ) [inline]
```

initialize from initialization list

5.1.2.4 `buffer()` [4/5]

```
buffer (
    goopax_device device,
    size_t size,
    T * host_ptr = nullptr,
    BUFFER_FLAGS flags = BUFFER_READ_WRITE ) [inline]
```

initialize with size

Parameters

<i>device</i>	device on which to allocate memory
<i>host_ptr</i>	pointer to host memory. If supplied, the buffer will be linked to the given address in host memory. <i>host_ptr</i> must point to a memory region that is at least <code>size*sizeof(T)</code> bytes long. The data may or may not be buffered during a kernel call, depending on the device driver. After the kernel has finished, the data is synchronized and can be accessed by the host.
<i>flags</i>	access flags

5.1.2.5 `buffer()` [5/5]

`buffer` () [default]

default constructor. The buffer cannot be used until replaced by a buffer that has been properly initialized.

5.1.3 Member Function Documentation

5.1.3.1 `copy()` [1/2]

```
void copy (
    const buffer< T, SRC_SIZE_TYPE > & src,
    typename enable_if< true, SRC_SIZE_TYPE >::type count,
    SIZE_TYPE start_dest,
    typename enable_if< true, SRC_SIZE_TYPE >::type start_src ) [inline]
```

copy data from one buffer to another buffer

Parameters

<i>src</i>	source buffer
<i>count</i>	number of elements to copy
<i>start_dest</i>	start position in destination buffer
<i>start_src</i>	start position in source buffer

5.1.3.2 `copy()` [2/2]

```
void copy (
    const buffer< T, SRC_SIZE_TYPE > & src ) [inline]
```

copy data from one buffer to another buffer

Parameters

<i>src</i>	source buffer
------------	---------------

5.1.3.3 `copy_from_host()` [1/2]

```
void copy_from_host (
    const T * p,
```

```

    size_t beginpos,
    size_t endpos ) [inline]

```

copy data from host memory

Parameters

<i>p</i>	pointer in host memory
<i>beginpos</i>	start position in device memory
<i>endpos</i>	next to last position in device memory

5.1.3.4 `copy_from_host()` [2/2]

```

void copy_from_host (
    const T * p ) [inline]

```

copy data from host

Parameters

<i>p</i>	pointer in host memory
----------	------------------------

5.1.3.5 `copy_to_host()` [1/2]

```

void copy_to_host (
    T * p,
    size_t beginpos,
    size_t endpos ) const [inline]

```

copy data to host

Parameters

<i>p</i>	pointer in host memory
<i>beginpos</i>	start position in device memory
<i>endpos</i>	next to last position in device memory

5.1.3.6 `copy_to_host()` [2/2]

```

void copy_to_host (
    T * p ) const [inline]

```

copy data to host

Parameters

<i>p</i>	pointer in host memory
----------	------------------------

5.1.3.7 `fill()` [1/2]

```
void fill (
    T value,
    size_type begin,
    size_type end ) [inline]
```

Fill buffer with value

Parameters

<i>value</i>	value to fill
<i>begin</i>	begin of fill region
<i>end</i>	next to last element to fill

5.1.3.8 `fill()` [2/2]

```
void fill (
    T value ) [inline]
```

Fill buffer with value

Parameters

<i>value</i>	value to fill
--------------	---------------

5.1.3.9 `get_device()`

```
goopax_device get_device ( ) const [inline]
```

Returns device on which the buffer is allocated.

5.1.3.10 `max()` [1/2]

```
T max (
    const size_type begin,
    const size_type end ) const [inline]
```

Finds maximum value

5.1.3.11 `max()` [2/2]

```
T max ( ) const [inline]
```

Finds maximum value

5.1.3.12 `min()` [1/2]

```
T min (
    const size_type begin,
    const size_type end ) const [inline]
```

Finds minimum value

5.1.3.13 `min()` [2/2]

```
T min ( ) const [inline]
```

Finds minimum value

5.1.3.14 `operator=()` [1/2]

```
buffer& operator= (
    const V & v ) [inline]
```

copy data from `std::vector`

5.1.3.15 `operator=()` [2/2]

```
buffer& operator= (
    initializer_list< T > il ) [inline]
```

assign data via initialization list

5.1.3.16 `size()`

```
size_type size ( ) const [inline]
```

Buffer size

5.1.3.17 `sum()` [1/2]

```
T sum (
    const size_type begin,
    const size_type end ) const [inline]
```

Calculates the sum of all values in the given range

5.1.3.18 sum() [2/2]

```
T sum ( ) const [inline]
```

Calculates the sum of all values

5.1.3.19 to_vector()

```
vector<value_type> to_vector ( ) const [inline]
```

returns a std::vector that holds a copy of the data

5.1.4 Friends And Related Function Documentation**5.1.4.1 operator<<**

```
ostream& operator<< (
    ostream & s,
    const buffer< T, SIZE_TYPE > & b ) [friend]
```

Output the buffer content.

5.1.4.2 reinterpret

```
TO reinterpret (
    const buffer< TFROM, SIZE_FROM > & from ) [friend]
```

Reinterpreting buffer value type

Changes the buffer value type without changing the data content. The returned buffer points to the same memory buffer as the original buffer.

Parameters

<i>TO</i>	destination type
<i>from</i>	source buffer

5.2 buffer_map< T, is_const > Class Template Reference

Inherits map_base.

Public Member Functions

- `template<typename EIF = void, typename = typename enable_if<lis_const, EIF>::type> value_type * begin ()`
- `template<typename EIF = void, typename = typename enable_if<lis_const, EIF>::type> value_type * end ()`
- `const value_type * begin () const`
- `const value_type * end () const`
- `template<typename EIF = void, typename = typename enable_if<lis_const, EIF>::type> value_type * data ()`
- `const value_type * data () const`
- `template<typename EIF = void, typename = typename enable_if<lis_const, EIF>::type> value_type & operator\[\] (size_t k)`
- `const value_type & operator\[\] (size_t k) const`
- `template<typename SIZE_TYPE, typename = typename enable_if<is_const, SIZE_TYPE>::type> buffer_map (const buffer< T, SIZE_TYPE > &buf, size_t begin, size_t end)`
- `template<typename SIZE_TYPE, typename = typename enable_if<is_const, SIZE_TYPE>::type> buffer_map (const buffer< T, SIZE_TYPE > &buf)`
- `template<typename SIZE_TYPE > buffer_map (buffer< T, SIZE_TYPE > &buf, size_t begin, size_t end)`
- `template<typename SIZE_TYPE > buffer_map (buffer< T, SIZE_TYPE > &buf)`

5.2.1 Detailed Description

```
template<typename T, bool is_const = false>
class goopax::impl::buffer_map< T, is_const >
```

buffer map. Can be used to access buffers from host code. For performance reasons, buffer maps should not be called within a loop, but before the loop, if possible. buffer maps must be out of scope before the buffer is used in a kernel call or in another operation.

Template Parameters

<i>T</i>	value type, must match the buffer value type
<i>is_const</i>	if true, allows mapping of const buffers

5.2.2 Constructor & Destructor Documentation

5.2.2.1 `buffer_map()` [1/4]

```
buffer_map (
    const buffer< T, SIZE_TYPE > & buf,
    size_t begin,
    size_t end ) [inline]
```

constructor. Maps the buffer to host memory.

Parameters

<i>buf</i>	buffer to map.
<i>begin</i>	first index
<i>end</i>	last+1 index

5.2.2.2 `buffer_map()` [2/4]

```
buffer_map (
    const buffer< T, SIZE_TYPE > & buf ) [inline]
```

constructor. Maps the buffer to host memory.

Parameters

<i>buf</i>	buffer to map.
------------	----------------

5.2.2.3 `buffer_map()` [3/4]

```
buffer_map (
    buffer< T, SIZE_TYPE > & buf,
    size_t begin,
    size_t end ) [inline]
```

constructor. Maps the buffer to host memory.

Parameters

<i>buf</i>	buffer to map.
<i>begin</i>	first index
<i>end</i>	last+1 index

5.2.2.4 `buffer_map()` [4/4]

```
buffer_map (
    buffer< T, SIZE_TYPE > & buf ) [inline]
```

constructor. Maps the buffer to host memory.

Parameters

<i>buf</i>	buffer to map.
------------	----------------

5.2.3 Member Function Documentation

5.2.3.1 `begin()` [1/2]

`value_type* begin () [inline]`

begin pointer, const

5.2.3.2 `begin()` [2/2]

`const value_type* begin () const [inline]`

begin pointer

5.2.3.3 `data()` [1/2]

`value_type* data () [inline]`

same as [begin\(\)](#)

5.2.3.4 `data()` [2/2]

`const value_type* data () const [inline]`

same as [begin\(\)](#)

5.2.3.5 `end()` [1/2]

`value_type* end () [inline]`

end pointer, const

5.2.3.6 `end()` [2/2]

`const value_type* end () const [inline]`

end pointer

5.2.3.7 `operator[]()` [1/2]

`value_type& operator[] (
 size_t k) [inline]`

accesses element k

Parameters

<i>k</i>	element to get reference to
----------	-----------------------------

5.2.3.8 operator[]() [2/2]

```
const value_type& operator[] (
    size_t k ) const [inline]
```

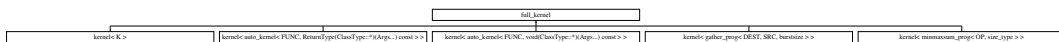
accesses element *k*

Parameters

<i>k</i>	element to get const reference to
----------	-----------------------------------

5.3 full_kernel Struct Reference

Inheritance diagram for full_kernel:



Public Member Functions

- void **force_compile** (goopax_device device)
force compilation of kernel. Only useful for performance measurements and other occasions.
- **full_kernel** (full_kernel &&b) noexcept
move semantics
- **full_kernel** & operator= (full_kernel &&b) noexcept
move semantics

overriding default thread numbers

- void **force_local_size** (goopax_device device, Tint local_size)
forces the local size for this device to the given value
- void **force_global_size** (goopax_device device, Tint global_size)
forces the global size for this device to the given value
- void **force_num_groups** (goopax_device device, Tint num_groups)
forces the number of workgroups for this device to the given value
- void **force_compfac** (goopax_device device, Tint compfac)

5.3.1 Detailed Description

kernel base

5.4 `gather_add< T >` Struct Template Reference

Inherits `gather_base< T >`.

Static Public Member Functions

- static `T` `init` ()
- `template<class U >`
static `U` `op` (const `U` &a, const `U` &b)

5.4.1 Detailed Description

```
template<class T>  
struct goopax::impl::gather_add< T >
```

returns the sum of all values.

5.4.2 Member Function Documentation

5.4.2.1 `init()`

```
static T init ( ) [inline], [static]
```

null element of the operation, used by `buffer::sum()` operation.

5.4.2.2 `op()`

```
static U op (  
    const U & a,  
    const U & b ) [inline], [static]
```

here the operation is defined.

5.5 `gather_max< T >` Struct Template Reference

Inherits `gather_base< T >`.

5.5.1 Detailed Description

```
template<class T>  
struct goopax::impl::gather_max< T >
```

returns the maximum of all values

5.6 gather_min< T > Struct Template Reference

Inherits gather_base< T >.

5.6.1 Detailed Description

```
template<class T>
struct goopax::impl::gather_min< T >
```

returns the minimum of all values

5.7 gettype< T, EIF > Struct Template Reference

type info and type conversion mechanism With the gettype struct, CPU and GPU types can be converted to each other, and type information obtained typename gettype<T>::cpu converts T to the corresponding CPU type. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. typename gettype<T>::gpu converts T to the corresponding GPU type. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. gettype<↔ T>::size provides the size of type T in bytes. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. typename gettype<T>::template change<U>::type changes the type of T to type U while preserving CPU/GPU/debug mode. If T is CPU type, the resulting type is also a CPU type. If T is GPU type, the resulting type is a GPU type. If T is debug data type, the resulting type is also a debug data type.

5.7.1 Detailed Description

```
template<typename T, typename EIF = void>
struct goopax::impl::gettype< T, EIF >
```

type info and type conversion mechanism With the gettype struct, CPU and GPU types can be converted to each other, and type information obtained typename gettype<T>::cpu converts T to the corresponding CPU type. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. typename gettype<T>::gpu converts T to the corresponding GPU type. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. gettype<↔ T>::size provides the size of type T in bytes. T can be CPU type, GPU type, or CPU debug type, or a goopax struct type. typename gettype<T>::template change<U>::type changes the type of T to type U while preserving CPU/GPU/debug mode. If T is CPU type, the resulting type is also a CPU type. If T is GPU type, the resulting type is a GPU type. If T is debug data type, the resulting type is also a debug data type.

5.8 goopax_device Class Reference

Goopax device. This class is used to reference the physical devices in the system.

Public Member Functions

- bool [valid](#) () const
checks if the device is valid. This will return false if the device is default-constructed. or if [default_device\(\)](#) did not find
- [envmode get_envmode](#) () const
returns the backend type that is used to access this device
- [size_t global_memory_total](#) () const
Total amount of global memory.
- [size_t global_memory_free](#) () const
Free global memmory, or [numeric_limits<size_t>::max\(\)](#) if unknown.
- [Tbool support_svm](#) () const
Returns true if the device supports shared memory addressing between the host and potentially other devices.
- void [wait_all](#) () const
Waits for all kernels that currently running or pending to finish on the device.
- [Tbool finished_all](#) () const
Tests if all kernels on the device have finished.
- bool [operator==](#) ([goopax_device](#) b) const
tests if two devices are the same
- bool [operator!=](#) ([goopax_device](#) b) const
tests if two devices are different
- [goopax_device & operator=](#) (const [goopax_device](#) &b)
- [goopax_device](#) (const [goopax_device](#) &b)

overriding default thread numbers

- void [force_local_size](#) (Tuint value)
forces the local size for this device to the given value
- void [force_global_size](#) (Tuint value)
forces the global size for this device to the given value
- void [force_num_groups](#) (Tuint value)
forces the number of workgroups for this device to the given value
- void [force_compfac](#) (Tuint value)

default thread numbers

These default values are used if they are not overridden

- Tuint [default_local_size](#) () const
local size
- Tuint [default_num_subthreads](#) () const
instruction level parallelism
- Tuint [default_num_groups](#) () const
number of thread groups
- Tuint [default_global_size](#) () const
*total number of threads. Equals $local_size * num_groups$*
- Tuint [default_local_mem](#) () const
Size of local memory.

floating point support

- [Tbool support_type](#) (half) const
Returns true if the device supports half precision.
- [Tbool support_type](#) (float) const
Returns true if the device supports single precision floating points.
- [Tbool support_type](#) (double) const

Returns true if the device supports double precision floating points.

denormalization support

- Tbool `support_fp_denorm` (half) const
Returns true if the device supports denormalized numbers for `gpu_half`.
- Tbool `support_fp_denorm` (float) const
Returns true if the device supports denormalized numbers for `gpu_float`.
- Tbool `support_fp_denorm` (double) const
Returns true if the device supports denormalized numbers for `gpu_double`.

device information

- const char * `vendor` () const
Vendor name.
- const char * `name` () const
device name
- double `max_frequency` () const
max device frequency
- double `flops` () const
Returns an estimate for the floating point operations the device can perform per second.
- Tuint `warp_size` () const
Returns the warp size of the device. This is typically the SIMD instruction width. Threads within a warp cannot branch into loops and if-clauses without all other threads waiting for them to finish.

5.8.1 Detailed Description

Goopax device. This class is used to reference the physical devices in the system.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 goopax_device()

```
goopax_device (
    const goopax_device & b ) [inline]
```

copy constructor

5.8.3 Member Function Documentation

5.8.3.1 operator=()

```
goopax_device& operator= (
    const goopax_device & b ) [inline]
```

copy operator

5.9 EX::goopax_exception Struct Reference

base exception thrown by goopax

Inherits exception.

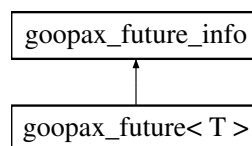
Inherited by EX::assertion, EX::backend, EX::general, EX::integrity, EX::memory, and EX::syntax.

5.9.1 Detailed Description

base exception thrown by goopax

5.10 goopax_future< T > Class Template Reference

Inheritance diagram for goopax_future< T >:



Public Member Functions

- void `wait ()`
wait for kernel to finish
- T `get ()`
wait for kernel to finish and get value
- `goopax_future ()`=default
default constructor
- `goopax_future (goopax_future &&)`=default
move semantics
- `goopax_future & operator= (goopax_future &&)`=default
move semantics

5.10.1 Detailed Description

```

template<typename T>
class goopax::impl::goopax_future< T >

```

future class for gather return values

Template Parameters

<code>T</code>	return type. May be a C++ intrinsic type, or a goopax struct type
----------------	---

5.10.2 Member Function Documentation

5.10.2.1 get()

```
T get ( ) [inline]
```

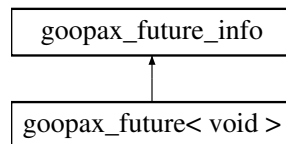
wait for kernel to finish and get value

Returns

value gathered from all threads and combined using the gather operation

5.11 goopax_future< void > Class Template Reference

Inheritance diagram for goopax_future< void >:



Public Member Functions

- void `wait ()`
wait for kernel to finish
- `goopax_future ()`=default
default constructor
- `goopax_future (goopax_future &&)`=default
move semantics
- `goopax_future & operator= (goopax_future &&)`=default
move semantics

5.11.1 Detailed Description

```
template<>
class goopax::impl::goopax_future< void >
```

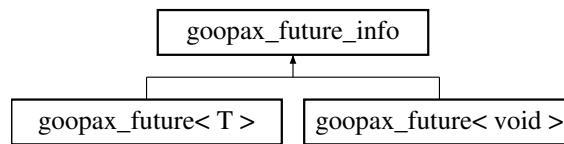
future class, Returned from kernels that don't return a value.

Template Parameters

<i>T</i>	return type. May be a C++ intrinsic type, or a goopax struct type
----------	---

5.12 goopax_future_info Class Reference

Inheritance diagram for goopax_future_info:



Public Member Functions

- bool [finished](#) () const
returns true if kernel has finished execution
- size_t [local_size](#) () const
number of threads per workgroup
- size_t [global_size](#) () const
total number of threads
- size_t [num_groups](#) () const
number of workgroups

5.12.1 Detailed Description

Common future class base, provides information about the kernel execution.

5.13 goopax_struct_type< std::array< T, N > > Struct Template Reference

std::array and derived classes

5.13.1 Detailed Description

```

template<typename T, size_t N>
struct goopax::goopax_struct_type< std::array< T, N > >
  
```

std::array and derived classes

5.14 gpu_ostream Class Reference

Public Member Functions

- [gpu_ostream](#) (ostream &os0)

Friends

- `gpu_ostream & operator<< (gpu_ostream &out, const string &s)`
output string
- `template<class T > gpu_ostream & operator<< (gpu_ostream &out, const gpu_type< T > &val)`
output gpu value
- `template<typename T , typename = typename enable_if<is_fundamental<T>::value || is_half<T>::value>::type> gpu_ostream & operator<< (gpu_ostream &s, T a)`
output cpu value
- `template<class V > gpu_ostream & operator<< (typename enable_if< is_container< V >::value, gpu_ostream >::type &out, const V &v)`
output STL vector
- `gpu_ostream & operator<< (gpu_ostream &out, const char *s)`
output C-style string
- `gpu_ostream & operator<< (gpu_ostream &s, ostream &(*__pf)(ostream &))`
output endl/flush
- `gpu_ostream & operator<< (gpu_ostream &s, decltype(&hex) h)`
modify stream, e.g. by hex or dec modifiers
- `template<class T , class TA > gpu_ostream & operator<< (gpu_ostream &out, const checknan< T, TA > &val)`
output cpu debug value

5.14.1 Detailed Description

Streaming class that can be used from within GPU kernels. The usage is similar to `std::ostream`. The output is written to the `std::ostream` object specified in the constructor. The output may be delayed until `wait()` has been called for the kernel/device. The order in which concurrent threads write is undefined. The output of any thread is only interrupted by other threads when 'endl' or 'flush' is used. If the output should not be interrupted, use "\n" instead of endl.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 `gpu_ostream()`

```
gpu_ostream (
    ostream & os0 ) [inline]
```

Constructor.

Parameters

<code>os0</code>	std::ostream object to which the output is passed.
------------------	--

5.15 gpu_type< T, scope > Class Template Reference

Inherits wrapper< TYPE, IMPL >.

Public Member Functions

- template<typename EA = void, typename EB = typename enable_if< true , EA>::type>
[gpu_bool operator==](#) (const [gpu_type](#) &b) const
Compare two GPU values for equality.
- template<typename EA = void, typename EB = typename enable_if< true , EA>::type>
[gpu_bool operator!=](#) (const [gpu_type](#) &b) const
Compare two GPU values for inequality.
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_bool operator<](#) (const [gpu_type](#) &b) const
Compare two GPU values.
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_bool operator>=](#) (const [gpu_type](#) &b) const
Compare two GPU values.
- template<typename EA = void, typename EB = typename enable_if< is_bool , EA>::type>
[gpu_bool operator &&](#) (const [gpu_type](#) &b) const
boolean AND
- template<typename EA = void, typename EB = typename enable_if< is_bool , EA>::type>
[gpu_bool operator||](#) (const [gpu_type](#) &b) const
boolean OR
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_type operator+](#) (const [gpu_type](#) &b) const
addition
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_type operator*](#) (const [gpu_type](#) &b) const
multiplication
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_type operator/](#) (const [gpu_type](#) &b) const
division
- template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type>
[gpu_type operator &](#) (const [gpu_type](#) &b) const
bitwise AND
- template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type>
[gpu_type operator|](#) (const [gpu_type](#) &b) const
bitwise OR
- template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type>
[gpu_type operator^](#) (const [gpu_type](#) &b) const
bitwise xor
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_type operator<<](#) (const [gpu_uint](#) &b) const
left shift
- template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type>
[gpu_type operator>>](#) (const [gpu_uint](#) &b) const
right shift
- template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type>
[gpu_type operator%](#) (const [gpu_type](#) &b) const
modulo operator

- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type operator- (const gpu_type &b) const`
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type operator- () const`
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type operator+ () const`
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_bool operator> (const gpu_type &b) const`
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_bool operator<= (const gpu_type &b) const`
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator++ ()`
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator-- ()`
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type operator++ (int)`
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type operator-- (int)`
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type & operator+= (const gpu_type &b)`
addition
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type & operator-= (const gpu_type &b)`
subtraction
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type & operator*= (const gpu_type &b)`
multiplication
- `template<typename EA = void, typename EB = typename enable_if< !is_bool , EA>::type> gpu_type & operator/= (const gpu_type &b)`
division
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator &= (const gpu_type &b)`
bitwise AND
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator |= (const gpu_type &b)`
bitwise OR
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator ^= (const gpu_type &b)`
bitwise XOR
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator <<= (const gpu_uint &b)`
left shift
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator >>= (const gpu_uint &b)`
right shift
- `template<typename EA = void, typename EB = typename enable_if< is_int , EA>::type> gpu_type & operator %= (const gpu_type &b)`
modulo
- `void gpu_break ()`
- `void gpu_continue ()`
- `gpu_type ()`
- `template<typename U > gpu_type (const struct wrapper< gpu_type< U >, gpu_writer_base_impl > &b)`
- `gpu_type (T d)`

- `template<typename U , typename = typename enable_if<is_half<T>::value && is_arithmetic<U>::value>::type>`
`gpu_type` (U d)
- `template<typename TA , typename BT >`
`gpu_type` (const checknan< BT, TA > &d)
- `template<typename B , typename FOO = typename enable_if<(((is_int && is_integral::value && !is_same<B, bool>::value) ||`
(is_floating_point<T>::value && is_floating_point::value)) && gettype<T>::size >= gettype::size>
`gpu_type` (const `gpu_type`< B > &b)
- `gpu_type` (const `gpu_type` &b)
- `gpu_type` & `operator=` (const `gpu_type` &b)

5.15.1 Detailed Description

```
template<typename T, memory::scope scope>
class goopax::impl::gpu_type< T, scope >
```

GPU data type, used in GPU kernels to store values in registers.

Template Parameters

<i>T</i>	corresponding CPU intrinsic type. Pointers are represented as <code>gpu_type<T*></code> or <code>gpu_type<const T*></code> , where T is a CPU intrinsic type or a goopax struct type.
----------	---

5.15.2 Constructor & Destructor Documentation

5.15.2.1 `gpu_type()` [1/7]

```
gpu_type ( ) [inline]
```

default constructor for uninitialized variable.

5.15.2.2 `gpu_type()` [2/7]

```
gpu_type (
    const struct wrapper< gpu_type< U >, gpu_writer_base_impl > & b ) [inline],
[explicit]
```

explicit type conversion

5.15.2.3 `gpu_type()` [3/7]

```
gpu_type (
    T d ) [inline]
```

conversion from CPU type

5.15.2.4 `gpu_type()` [4/7]

```
gpu_type (
    U d ) [inline]
```

conversion from CPU type

5.15.2.5 `gpu_type()` [5/7]

```
gpu_type (
    const checknan< BT, TA > & d ) [inline]
```

conversion from CPU debug type

5.15.2.6 `gpu_type()` [6/7]

```
gpu_type (
    const gpu_type< B > & b ) [inline]
```

Implicit type conversion The type conversion rules are stricter than the standard C/C++ type conversion rules to avoid unintended performance loss: Integer values are only implicitly converted to integers of at least the same size or to floating point numbers. floating point numbers are only implicitly converted to other floating point numbers of at least the same size. Other conversions need to be done explicitly, with `static_cast`, or C/C++ style cast.

5.15.2.7 `gpu_type()` [7/7]

```
gpu_type (
    const gpu_type< T, scope > & b ) [inline]
```

copy constructor

5.15.3 Member Function Documentation

5.15.3.1 `gpu_break()`

```
void gpu_break ( ) [inline]
```

Break from loop. Only valid on loop variables. Calling `gpu_break()` on non-loop variables results in an error.

5.15.3.2 `gpu_continue()`

```
void gpu_continue ( ) [inline]
```

Continue loop. Only valid on loop variables. Calling `gpu_continue()` on non-loop variables results in an error.

5.15.3.3 operator+()

```
gpu_type operator+ ( ) const [inline]
```

unary +

5.15.3.4 operator++() [1/2]

```
gpu_type& operator++ ( ) [inline]
```

increment

5.15.3.5 operator++() [2/2]

```
gpu_type operator++ (
    int ) [inline]
```

increment

5.15.3.6 operator-() [1/2]

```
gpu_type operator- (
    const gpu_type< T, scope > & b ) const [inline]
```

subtract

5.15.3.7 operator-() [2/2]

```
gpu_type operator- ( ) const [inline]
```

unary -

5.15.3.8 operator--() [1/2]

```
gpu_type& operator-- ( ) [inline]
```

decrement

5.15.3.9 operator--() [2/2]

```
gpu_type operator-- (
    int ) [inline]
```

decrement

5.15.3.10 operator<=()

```
gpu_bool operator<= (
    const gpu_type< T, scope > & b ) const [inline]
```

comparison

5.15.3.11 operator=()

```
gpu_type& operator= (
    const gpu_type< T, scope > & b ) [inline]
```

assignment operator

5.15.3.12 operator>()

```
gpu_bool operator> (
    const gpu_type< T, scope > & b ) const [inline]
```

comparison

5.16 image_array_buffer< DIM, T, normalized > Class Template Reference

Inherits image_buffer_base< DIM, IS_ARRAY, T, normalized >.

Classes

- struct [image_slice](#)
Reference to single image in the image array.

Public Member Functions

- size_type [array_size](#) () const
- [image_slice](#)< false > [operator\[\]](#) (size_type imagenum)
- [image_slice](#)< true > [operator\[\]](#) (size_type imagenum) const
- void [copy](#) (const [image_array_buffer](#) &b, array< size_type, DIM+1 > src_origin, array< size_type, DIM+1 > dest_origin, array< size_type, DIM+1 > region)
- void [copy](#) (const [image_array_buffer](#) &b)
- void [copy_to_host](#) (T *p, array< size_type, DIM+1 > origin, array< size_type, DIM+1 > region, array< size_type, DIM > pitchdim=makearray< size_type, DIM >(0)) const
- void [copy_to_host](#) (T *p) const
- vector< value_type > [to_vector](#) () const
- void [copy_from_host](#) (const T *p, array< size_type, DIM+1 > origin, array< size_type, DIM+1 > region, array< size_type, DIM > pitchdim=makearray< size_type, DIM >(0))
- [image_array_buffer](#) ()
- [image_array_buffer](#) ([goopax_device](#) device, size_type num_images, const array< size_type, DIM > size, [IMAGE_FLAGS](#) flags=IMAGE_READ_WRITE, value_type *host_ptr=nullptr, array< size_type, DIM > pitchdim=makearray< size_type, DIM >(0))
- void [fill](#) (T value, array< size_type, DIM+1 > origin, array< size_type, DIM+1 > region)
- void [fill](#) (T value)

Static Public Member Functions

- static `image_array_buffer create_from_gl` (`goopax_device` device, unsigned int GLres, `IMAGE_FLAGS` flags=`IMAGE_READ_WRITE`, unsigned int GLtarget=(`DIM==1` ? `0x0DE0` :(`DIM==2` ? `0x0DE1` :`0x806F`)), int miplevel=0)
- static `image_array_buffer create_from_cl` (`goopax_device` device, `_cl_mem` *mem)

5.16.1 Detailed Description

```
template<unsigned int DIM, typename T, bool normalized>
class goopax::impl::image_array_buffer< DIM, T, normalized >
```

/ Image array. The image will be allocated on the device.

Template Parameters

<i>DIM</i>	Number of dimensions. Must be 1 or 2.
<i>T</i>	Element data type. Must be <code>std::array<C, N></code> , or other goopax struct of identical types. C: channel type (<code>int32_t</code> , <code>int16_t</code> , <code>int8_t</code> , <code>uint32_t</code> , <code>uint16_t</code> , <code>uint8_t</code> , <code>float</code> , <code>half</code>), N: number of channels (must be 4 at the moment).
<i>normalized</i>	if set to true, image values are normalized to floats in the range 0..1 for unsigned channel types, or -1..1 for signed channel types. Defaults to false if the channel data type allows unnormalized access

5.16.2 Constructor & Destructor Documentation

5.16.2.1 image_array_buffer() [1/2]

```
image_array_buffer ( ) [inline]
```

default constructor. Image must not be used until replaced by a real image.

5.16.2.2 image_array_buffer() [2/2]

```
image_array_buffer (
    goopax_device device,
    size_type num_images,
    const array< size_type, DIM > size,
    IMAGE_FLAGS flags = IMAGE_READ_WRITE,
    value_type * host_ptr = nullptr,
    array< size_type, DIM > pitchdim = makearray<size_type, DIM>(0) ) [inline]
```

Allocated image array on the device.

Parameters

<i>device</i>	device to allocate image on
<i>num_devices</i>	number of images in the array
<i>size</i>	image dimensions
<i>flags</i>	must be one of IMAGE_READ_WRITE, IMAGE_READ_ONLY, IMAGE_WRITE_ONLY, IMAGE_WRITE_DISCARD
<i>host_ptr</i>	if used, image array will be linked to the specified address in host memory.
<i>pitchdim</i>	pitch size in pixels

5.16.3 Member Function Documentation

5.16.3.1 array_size()

```
size_type array_size ( ) const [inline]
```

Number of images in the array

5.16.3.2 copy() [1/2]

```
void copy (
    const image_array_buffer< DIM, T, normalized > & b,
    array< size_type, DIM+1 > src_origin,
    array< size_type, DIM+1 > dest_origin,
    array< size_type, DIM+1 > region ) [inline]
```

copy image array The last indices in `src_origin`, `dest_origin`, and `region` refer to the image slice, i.e., for 1D image arrays, coordinates in the form {x, imagenum} are passed, for 2D image arrays, coordinates in the form {x, y, imagenum} are passed.

Parameters

<i>b</i>	source image array
<i>src_origin</i>	offset in source image array
<i>dest_origin</i>	offset in destination image array
<i>region</i>	size of image array patch.

5.16.3.3 copy() [2/2]

```
void copy (
    const image_array_buffer< DIM, T, normalized > & b ) [inline]
```

copy image array

5.16.3.4 copy_from_host()

```
void copy_from_host (
    const T * p,
    array< size_type, DIM+1 > origin,
    array< size_type, DIM+1 > region,
    array< size_type, DIM > pitchdim = makearray<size_type, DIM>(0) ) [inline]
```

copy image array from host memory

Parameters

<i>p</i>	pointer in host memory
<i>origin</i>	image offset in device
<i>region</i>	size of image patch
<i>pitchdim</i>	full image dimensions in host memory. If filled with zeros, will be calculated from image size

5.16.3.5 copy_to_host() [1/2]

```
void copy_to_host (
    T * p,
    array< size_type, DIM+1 > origin,
    array< size_type, DIM+1 > region,
    array< size_type, DIM > pitchdim = makearray<size_type, DIM>(0) ) const [inline]
```

copy image array to host memory.

Parameters

<i>p</i>	pointer to host memory
<i>origin</i>	offset in device image
<i>region</i>	size of image patch
<i>pitchdim</i>	full image dimensions in host memory. If filled with zeros, will be calculated from image size

5.16.3.6 copy_to_host() [2/2]

```
void copy_to_host (
    T * p ) const [inline]
```

copy image array to host memory

Parameters

<i>p</i>	pointer to host memory
----------	------------------------

5.16.3.7 fill() [1/2]

```
void fill (
    T value,
    array< size_type, DIM+1 > origin,
    array< size_type, DIM+1 > region ) [inline]
```

Fill image array

Parameters

<i>value</i>	color to fill
<i>origin</i>	origin in destination image. The last value specifies the image number in the array.
<i>region</i>	size of image patch to fill. The last value is the number of images to fill in the array.

5.16.3.8 fill() [2/2]

```
void fill (
    T value ) [inline]
```

Fill image array

Parameters

<i>value</i>	color to fill
--------------	---------------

5.16.3.9 operator[]() [1/2]

```
image_slice<false> operator[] (
    size_type imagenum ) [inline]
```

selects one image of the image array

5.16.3.10 operator[]() [2/2]

```
image_slice<true> operator[] (
    size_type imagenum ) const [inline]
```

selects one image of the image array

5.16.3.11 to_vector()

```
vector<value_type> to_vector ( ) const [inline]
```

returns a std::vector that holds a copy of the data

5.17 image_array_buffer_map< DIM, T, is_const > Class Template Reference

Inherits map_base.

Public Member Functions

- array< Tuint, DIM > [dimensions](#) () const
- Tsize_t [array_size](#) () const
- [image_buffer_map](#)< DIM, T, is_const > [operator\[\]](#) (Tsize_t slice)
- [image_buffer_map](#)< DIM, T, true > [operator\[\]](#) (Tsize_t slice) const
- template<bool normalized, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> [image_array_buffer_map](#) ([image_array_buffer](#)< DIM, T, normalized > &buf)
- template<bool normalized, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> [image_array_buffer_map](#) ([image_array_buffer](#)< DIM, T, normalized > &buf, array< Tuint, DIM+1 > origin, array< Tuint, DIM+1 > region)
- template<bool normalized, typename EIF = void, typename = typename enable_if<is_const, EIF>::type> [image_array_buffer_map](#) (const [image_array_buffer](#)< DIM, T, normalized > &buf)
- template<bool normalized, typename EIF = void, typename = typename enable_if<is_const, EIF>::type> [image_array_buffer_map](#) (const [image_array_buffer](#)< DIM, T, normalized > &buf, array< Tuint, DIM+1 > origin, array< Tuint, DIM+1 > region)

5.17.1 Detailed Description

```
template<unsigned int DIM, typename T, bool is_const = false>
class goopax::impl::image_array_buffer_map< DIM, T, is_const >
```

Maps an image buffer array to host memory

5.17.2 Constructor & Destructor Documentation

5.17.2.1 image_array_buffer_map() [1/4]

```
image_array_buffer_map (
    image_array_buffer< DIM, T, normalized > & buf ) [inline]
```

constructor.

Parameters

<i>buf</i>	image array that is to be mapped to host memory
------------	---

5.17.2.2 `image_array_buffer_map()` [2/4]

```
image_array_buffer_map (
    image_array_buffer< DIM, T, normalized > & buf,
    array< Tuint, DIM+1 > origin,
    array< Tuint, DIM+1 > region ) [inline]
```

constructor.

Parameters

<i>buf</i>	image array that is to be mapped to host memory
<i>origin</i>	of image region. The last element is the first array index
<i>region</i>	size of image region. The last element specifies the number of images.

5.17.2.3 `image_array_buffer_map()` [3/4]

```
image_array_buffer_map (
    const image_array_buffer< DIM, T, normalized > & buf ) [inline]
```

constructor.

Parameters

<i>buf</i>	image array that is to be mapped to host memory
------------	---

5.17.2.4 `image_array_buffer_map()` [4/4]

```
image_array_buffer_map (
    const image_array_buffer< DIM, T, normalized > & buf,
    array< Tuint, DIM+1 > origin,
    array< Tuint, DIM+1 > region ) [inline]
```

constructor.

Parameters

<i>buf</i>	image array that is to be mapped to host memory
<i>origin</i>	of image region. The last element is the first array index
<i>region</i>	size of image region. The last element specifies the number of images.

5.17.3 Member Function Documentation

5.17.3.1 array_size()

```
Tsize_t array_size ( ) const [inline]
```

number of images in the array map

5.17.3.2 dimensions()

```
array<Tuint, DIM> dimensions ( ) const [inline]
```

image dimensions of the image map.

5.17.3.3 operator[]() [1/2]

```
image_buffer_map<DIM, T, is_const> operator[] (
    Tsize_t slice ) [inline]
```

accessing image slice in the image array

Parameters

<i>slice</i>	image number in the array
--------------	---------------------------

Returns

image map to the image

5.17.3.4 operator[]() [2/2]

```
image_buffer_map<DIM, T, true> operator[] (
    Tsize_t slice ) const [inline]
```

accessing image slice in the image array

Parameters

<i>slice</i>	image number in the array
--------------	---------------------------

Returns

image map to the image

5.18 image_array_resource< DIM, T, normalized > Class Template Reference

Inherits image_resource_base< DIM, IS_ARRAY, T, normalized >.

Classes

- struct [image_slice](#)
Reference to single image in the image array.

Public Member Functions

- [image_slice](#)< [image_array_resource](#) & > [operator\[\]](#) (size_type imagenum)
- [image_slice](#)< const [image_array_resource](#) & > [operator\[\]](#) (size_type imagenum) const
- [image_array_resource](#) (const [buffer_type](#) &tex)

5.18.1 Detailed Description

```
template<unsigned int DIM, typename T, bool normalized>
class goopax::impl::image_array_resource< DIM, T, normalized >
```

Image array resources. Used for accessing an [image_array_buffer](#) from the kernel code.

Template Parameters

<i>DIM</i>	Number of dimensions. Must be 1 or 2.
<i>T</i>	Element data type. Must be <code>std::array<C, N></code> , or other goopax struct of identical types. C: channel type (<code>int32_t</code> , <code>int16_t</code> , <code>int8_t</code> , <code>uint32_t</code> , <code>uint16_t</code> , <code>uint8_t</code> , <code>float</code> , <code>half</code>), N: number of channels (must be 4 at the moment).
<i>normalized</i>	if set to true, image values are normalized to floats in the range 0..1 for unsigned channel types, or -1..1 for signed channel types. Defaults to false if the channel data type allows unnormalized access

5.18.2 Constructor & Destructor Documentation**5.18.2.1 image_array_resource()**

```
image_array_resource (
    const buffer\_type & tex ) [inline]
```

constructor.

Parameters

<i>tex</i>	<code>image_array_buffer</code> to link to. The template parameters of the <code>image_array_buffer</code> and the <code>image_array_resource</code> must match
------------	---

5.18.3 Member Function Documentation

5.18.3.1 `operator[]()` [1/2]

```
image_slice<image_array_resource&> operator[] (
    size_type imagenum ) [inline]
```

access image slice

Parameters

<i>imagenum</i>	image number in the array
-----------------	---------------------------

Returns

image slice

5.18.3.2 `operator[]()` [2/2]

```
image_slice<const image_array_resource&> operator[] (
    size_type imagenum ) const [inline]
```

access image slice (const)

Parameters

<i>imagenum</i>	image number in the array
-----------------	---------------------------

Returns

image slice

5.19 `image_buffer< DIM, T, normalized >` Class Template ReferenceInherits `image_buffer_base< DIM, IS_ARRAY, T, normalized >`.

Public Types

- using `map_type` = `image_buffer_map`< DIM, T >
< *pixe element type*

Public Member Functions

- `goopax_device` `get_device` () const
< *returns the device that the image is allocated on*
- void `copy` (const `image_buffer` &b, array< size_type, DIM > src_origin, array< size_type, DIM > dest_origin, array< size_type, DIM > region)
- void `copy` (const `image_buffer` &b)
- vector< value_type > `to_vector` () const
- void `copy_from_host` (const T *p, array< size_type, DIM > origin, array< size_type, DIM > region, array< size_type, DIM - 1 > pitchdim=makearray< size_type, DIM - 1 >(0))
- void `copy_from_host` (const T *p)
- void `fill` (T value, array< size_type, DIM > origin, array< size_type, DIM > region)
- void `fill` (T value)
- `image_buffer` (`goopax_device` device, const array< size_type, DIM > size, `IMAGE_FLAGS` flags=IMAGE_READ_WRITE, value_type *host_ptr=nullptr, array< size_type, DIM - 1 > pitchdim=makearray< size_type, DIM - 1 >(0))
- `image_buffer` ()
- void `copy_to_host` (T *p, array< size_type, DIM > origin, array< size_type, DIM > region, array< size_type, DIM - 1 > pitchdim=makearray< size_type, DIM - 1 >(0)) const

Static Public Member Functions

- static `image_buffer` `create_from_gl` (`goopax_device` device, unsigned int GLres, `IMAGE_FLAGS` flags=IMAGE_READ_WRITE, unsigned int GLtarget=(DIM==1 ? 0x0DE0 :(DIM==2 ? 0x0DE1 :0x806F)), int miplevel=0)
- static `image_buffer` `create_from_cl` (`goopax_device` device, `_cl_mem` *mem)

5.19.1 Detailed Description

```
template<unsigned int DIM, typename T, bool normalized>
class goopax::impl::image_buffer< DIM, T, normalized >
```

/ Image container. The image will be allocated on the device.

Template Parameters

<i>DIM</i>	Number of dimensions. Must be 1, 2, od 3.
<i>T</i>	Element data type. Must be <code>std::array<C, N></code> , or other goopax struct of identical types. C: channel type (<code>int32_t</code> , <code>int16_t</code> , <code>int8_t</code> , <code>uint32_t</code> , <code>uint16_t</code> , <code>uint8_t</code> , <code>float</code> , <code>half</code>), N: number of channels (must be 4 at the moment).
<i>normalized</i>	if set to true, image values are normalized to floats in the range 0..1 for unsigned channel types, or -1..1 for signed channel types. Defaults to false if the channel data type allows unnormalized access

5.19.2 Constructor & Destructor Documentation

5.19.2.1 image_buffer() [1/2]

```
image_buffer (
    goopax_device device,
    const array< size_type, DIM > size,
    IMAGE_FLAGS flags = IMAGE_READ_WRITE,
    value_type * host_ptr = nullptr,
    array< size_type, DIM - 1 > pitchdim = makearray<size_type, DIM - 1>(0) ) [inline]
```

Allocated image on the device.

Parameters

<i>device</i>	device to allocate image on
<i>size</i>	size of image
<i>flags</i>	must be one of IMAGE_READ_WRITE, IMAGE_READ_ONLY, IMAGE_WRITE_ONLY, IMAGE_WRITE_DISCARD
<i>host_ptr</i>	if used, image will be linked to the specified address in host memory.
<i>pitchdim</i>	pitch size in pixels

5.19.2.2 image_buffer() [2/2]

```
image_buffer ( ) [inline]
```

default constructor. Image must not be used until replaced by a real image.

5.19.3 Member Function Documentation

5.19.3.1 copy() [1/2]

```
void copy (
    const image_buffer< DIM, T, normalized > & b,
    array< size_type, DIM > src_origin,
    array< size_type, DIM > dest_origin,
    array< size_type, DIM > region ) [inline]
```

copy image region from another image of the same type.

Parameters

<i>b</i>	source image
<i>src_origin</i>	origin in source image
<i>dest_origin</i>	origin in *this
<i>region</i>	size of image region to copy

5.19.3.2 `copy()` [2/2]

```
void copy (
    const image_buffer< DIM, T, normalized > & b ) [inline]
```

copy image from another image with the same type and dimensions.

5.19.3.3 `copy_from_host()` [1/2]

```
void copy_from_host (
    const T * p,
    array< size_type, DIM > origin,
    array< size_type, DIM > region,
    array< size_type, DIM - 1 > pitchdim = makearray<size_type, DIM - 1>(0) ) [inline]
```

copy image region from host.

Parameters

<i>p</i>	source address in host memory
<i>origin</i>	origin of image in device
<i>region</i>	size of image region
<i>pitchdim</i>	row pitch dimensions (2D images) or row pitch+slice pitch dimensions (3D images), in pixels. If values are 0, the pitch is calculated from region

5.19.3.4 `copy_from_host()` [2/2]

```
void copy_from_host (
    const T * p ) [inline]
```

copy whole image from host.

Parameters

<i>p</i>	source address in host memory
----------	-------------------------------

5.19.3.5 copy_to_host()

```
void copy_to_host (
    T * p,
    array< size_type, DIM > origin,
    array< size_type, DIM > region,
    array< size_type, DIM - 1 > pitchdim = makearray<size_type, DIM - 1>(0) ) const
[inline]
```

copy image region to host.

Parameters

<i>p</i>	destination address in host memory
<i>origin</i>	origin of image in device
<i>region</i>	size of image region
<i>pitchdim</i>	row pitch dimensions (2D images) or row pitch+slice pitch dimensions (3D images), in pixels.

5.19.3.6 fill() [1/2]

```
void fill (
    T value,
    array< size_type, DIM > origin,
    array< size_type, DIM > region ) [inline]
```

Fill image array

Parameters

<i>value</i>	color to fill
<i>origin</i>	origin in destination image.
<i>region</i>	size of image patch to fill.

5.19.3.7 fill() [2/2]

```
void fill (
    T value ) [inline]
```

Fill image array

Parameters

<i>value</i>	color to fill
--------------	---------------

5.19.3.8 to_vector()

```
vector<value_type> to_vector ( ) const [inline]
```

returns a std::vector that holds a copy of the data

5.20 image_buffer_map< DIM, T, is_const > Class Template Reference

Inherits map_base.

Public Member Functions

- array< Tuint, DIM > [dimensions](#) () const
- template<typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> value_type & [operator\[\]](#) (array< Tuint, DIM > pos)
- const value_type & [operator\[\]](#) (array< Tuint, DIM > pos) const
- template<bool normalized, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> [image_buffer_map](#) (image_buffer< DIM, T, normalized > &buf)
- template<bool normalized, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> [image_buffer_map](#) (image_buffer< DIM, T, normalized > &buf, array< Tuint, DIM > origin, array< Tuint, DIM > region)
- template<bool normalized, typename EIF = void, typename = typename enable_if<is_const, EIF>::type> [image_buffer_map](#) (const image_buffer< DIM, T, normalized > &buf)
- template<bool normalized, typename EIF = void, typename = typename enable_if<is_const, EIF>::type> [image_buffer_map](#) (const image_buffer< DIM, T, normalized > &buf, array< Tuint, DIM > origin, array< Tuint, DIM > region)

5.20.1 Detailed Description

```
template<unsigned int DIM, typename T, bool is_const = false>
class goopax::impl::image_buffer_map< DIM, T, is_const >
```

Maps an image buffer to host memory and provides access to the image from the host.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 image_buffer_map() [1/4]

```
image_buffer_map (
    image_buffer< DIM, T, normalized > & buf ) [inline]
```

constructor.

Parameters

<i>buf</i>	image that is to be mapped to host memory
------------	---

5.20.2.2 image_buffer_map() [2/4]

```
image_buffer_map (
    image_buffer< DIM, T, normalized > & buf,
    array< Tuint, DIM > origin,
    array< Tuint, DIM > region ) [inline]
```

constructor.

Parameters

<i>buf</i>	image that is to be mapped to host memory
<i>origin</i>	begin of image region
<i>region</i>	size of image region

5.20.2.3 image_buffer_map() [3/4]

```
image_buffer_map (
    const image_buffer< DIM, T, normalized > & buf ) [inline]
```

constructor.

Parameters

<i>buf</i>	const image that is to be mapped to host memory.
------------	--

5.20.2.4 image_buffer_map() [4/4]

```
image_buffer_map (
    const image_buffer< DIM, T, normalized > & buf,
    array< Tuint, DIM > origin,
    array< Tuint, DIM > region ) [inline]
```

constructor.

Parameters

<i>buf</i>	image that is to be mapped to host memory
<i>origin</i>	begin of image region
<i>region</i>	size of image region

5.20.3 Member Function Documentation

5.20.3.1 dimensions()

```
array<Tuint, DIM> dimensions ( ) const [inline]
```

dimensions of image map

5.20.3.2 operator[]() [1/2]

```
value_type& operator[] (
    array< Tuint, DIM > pos ) [inline]
```

image access from host

Parameters

<i>pos</i>	position in image
------------	-------------------

Returns

reference to image pixel

5.20.3.3 operator[]() [2/2]

```
const value_type& operator[] (
    array< Tuint, DIM > pos ) const [inline]
```

image access from host

Parameters

<i>pos</i>	position in image
------------	-------------------

Returns

reference to image pixel

5.21 image_resource< DIM, T, normalized > Class Template Reference

Inherits image_resource_base< DIM, IS_ARRAY, T, normalized >.

Public Member Functions

- access_type [read](#) (array< size_type, DIM > pos) const
- template<typename U , typename = typename enable_if<allow_sampler, U>::type>
access_type [read](#) (array< U, DIM > pos, sampler_mode sampler) const
- template<typename POS , typename = typename enable_if<is_same<typename POS::value_type, size_type>::value>::type>
access_type [read](#) (POS pos) const
- template<typename POS , typename = typename enable_if<allow_sampler, POS>::type>
access_type [read](#) (POS pos, sampler_mode sampler) const
- void [write](#) (array< size_type, DIM > pos, access_type colors)
- template<typename POS , typename = typename enable_if<is_same<typename POS::value_type, size_type>::value>::type>
void [write](#) (POS pos, access_type colors)
- [image_resource](#) (buffer_type &tex)

Static Public Attributes

- static constexpr bool [allow_sampler](#) = base_t::allow_sampler(normalized)
true if samplers are allowed for the channel data type.

5.21.1 Detailed Description

```
template<unsigned int DIM, typename T, bool normalized>
class goopax::impl::image_resource< DIM, T, normalized >
```

Image resources. Used for accessing an [image_buffer](#) from the kernel code.

Template Parameters

<i>DIM</i>	Number of dimensions. Must be 1, 2, or 3.
<i>T</i>	Element data type. Must be std::array<C, N>, or other goopax struct of identical types. C: channel type (int32_t, int16_t, int8_t, uint32_t, uint16_t, uint8_t, float, half), N: number of channels (must be 4 at the moment).
<i>normalized</i>	if set to true, image values are normalized to floats in the range 0..1 for unsigned channel types, or -1..1 for signed channel types. Defaults to false if the channel data type allows unnormalized access

5.21.2 Constructor & Destructor Documentation

5.21.2.1 image_resource()

```
image_resource (
    buffer_type & tex ) [inline]
```

constructor.

Parameters

<i>tex</i>	image_buffer to link to. The template parameters of the image_buffer and the image_resource must match
------------	--

5.21.3 Member Function Documentation

5.21.3.1 `read()` [1/4]

```
access_type read (
    array< size_type, DIM > pos ) const [inline]
```

read from image

Parameters

<i>pos</i>	image position
<i>return</i>	pixel at the given position

5.21.3.2 `read()` [2/4]

```
access_type read (
    array< U, DIM > pos,
    sampler_mode sampler ) const [inline]
```

read from image with sampler

Parameters

<i>pos</i>	position, may be an array of type <code>gpu_int</code> , <code>gpu_uint</code> , or <code>gpu_float</code>
<i>sampler</i>	mode.
<i>return</i>	pixel

5.21.3.3 `read()` [3/4]

```
access_type read (
    POS pos ) const [inline]
```

read pixel.

Parameters

<i>pos</i>	pixel position. May be any container that provides the [] operator
<i>return</i>	pixel at the given position

5.21.3.4 read() [4/4]

```
access_type read (
    POS pos,
    sampler_mode sampler ) const [inline]
```

read from image with sampler

Parameters

<i>pos</i>	position, may be any container that provides the [] operator, and that holds values of type gpu_int, gpu_uint, or gpu_float
<i>sampler</i>	mode.
<i>return</i>	pixel

5.21.3.5 write() [1/2]

```
void write (
    array< size_type, DIM > pos,
    access_type colors ) [inline]
```

write to image

Parameters

<i>pos</i>	pixel position
<i>colors</i>	pixel color to write

5.21.3.6 write() [2/2]

```
void write (
    POS pos,
    access_type colors ) [inline]
```

write to image

Parameters

<i>pos</i>	pixel position
<i>colors</i>	pixel color to write. May be any container that provides the [] operator

5.22 image_array_buffer< DIM, T, normalized >::image_slice< is_const > Struct Template Reference

Reference to single image in the image array.

Public Member Functions

- array< size_type, DIM > [dimensions](#) () const
- template<bool C2, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> void [copy](#) (const [image_slice](#)< C2 > &b, array< size_type, DIM > src_origin, array< size_type, DIM > dest_origin, array< size_type, DIM > region)
- template<bool C2, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> void [copy](#) (const [image_slice](#)< C2 > &b)
- void [copy_to_host](#) (T *p, array< size_type, DIM > origin, array< size_type, DIM > region, size_type row_↔ pitch=0) const
- void [copy_to_host](#) (T *p) const
- vector< value_type > [to_vector](#) () const
- template<typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> void [copy_from_host](#) (const T *p, array< size_type, DIM > origin, array< size_type, DIM > region, size_type row_pitch=0)
- template<typename EIF = void, typename = typename enable_if<!is_const, EIF>::type> void [copy_from_host](#) (const T *p)

5.22.1 Detailed Description

```
template<unsigned int DIM, typename T, bool normalized>
template<bool is_const>
struct goopax::impl::image_array_buffer< DIM, T, normalized >::image_slice< is_const >
```

Reference to single image in the image array.

5.22.2 Member Function Documentation

5.22.2.1 copy() [1/2]

```
void copy (
    const image\_slice< C2 > & b,
    array< size_type, DIM > src_origin,
    array< size_type, DIM > dest_origin,
    array< size_type, DIM > region ) [inline]
```

copy from other image slice

Parameters

<i>src_origin</i>	offset in source image
<i>dest_origin</i>	offset in destination image
<i>region</i>	size of image patch to copy

5.22.2.2 copy() [2/2]

```
void copy (
    const image_slice< C2 > & b ) [inline]
```

copy from other image size

5.22.2.3 copy_from_host() [1/2]

```
void copy_from_host (
    const T * p,
    array< size_type, DIM > origin,
    array< size_type, DIM > region,
    size_type row_pitch = 0 ) [inline]
```

copy image slice from host memory

Parameters

<i>p</i>	pointer in host memory
<i>origin</i>	offset in device memory
<i>region</i>	size of image patch
<i>row_pitch</i>	pitch in host memory in pixels. Defaults to region[0]

5.22.2.4 copy_from_host() [2/2]

```
void copy_from_host (
    const T * p ) [inline]
```

copy image slice from host memoryx

5.22.2.5 copy_to_host() [1/2]

```
void copy_to_host (
    T * p,
    array< size_type, DIM > origin,
    array< size_type, DIM > region,
    size_type row_pitch = 0 ) const [inline]
```

Copy image slice to host memory.

Parameters

<i>p</i>	pointer to host memory
<i>origin</i>	offset in device
<i>region</i>	size of image patch
<i>row_pitch</i>	pitch in host memory in pixels. Defaults to region[0]

5.22.2.6 `copy_to_host()` [2/2]

```
void copy_to_host (
    T * p ) const [inline]
```

Copy image slice to host memory

5.22.2.7 `dimensions()`

```
array<size_type, DIM> dimensions ( ) const [inline]
```

image dimensions

5.22.2.8 `to_vector()`

```
vector<value_type> to_vector ( ) const [inline]
```

returns a `std::vector` that holds a copy of the data

5.23 `image_array_resource< DIM, T, normalized >::image_slice< RESREF >` Struct Template Reference

Reference to single image in the image array.

Public Member Functions

- `access_type read` (`array< size_type, DIM > pos`) `const`
- `template<typename U , typename = typename enable_if<allow_sampler, U>::type>`
`access_type read` (`array< U, DIM > pos, sampler_mode sampler=address_none|filter_nearest`) `const`
- `template<typename POS , typename = typename enable_if<is_same<typename POS::value_type, size_type>::value>::type>`
`access_type read` (`POS pos`) `const`
- `template<typename POS , typename = typename enable_if<allow_sampler, POS>::type>`
`access_type read` (`POS pos, sampler_mode sampler`) `const`
- `template<typename = void, typename = typename enable_if<is_same<RESREF, typename remove_const<RESREF>::type>↔::value>::type>`
`void write` (`array< size_type, DIM > pos, access_type colors`)
- `template<typename POS , typename = typename enable_if<is_same<typename POS::value_type, size_type>::value>::type>`
`void write` (`POS pos, access_type colors`)

5.23.1 Detailed Description

```
template<unsigned int DIM, typename T, bool normalized>
template<typename RESREF>
struct goopax::impl::image_array_resource< DIM, T, normalized >::image_slice< RESREF >
```

Reference to single image in the image array.

5.23.2 Member Function Documentation

5.23.2.1 read() [1/4]

```
access_type read (
    array< size_type, DIM > pos ) const [inline]
```

read from image slice

Parameters

<i>pos</i>	image position
<i>return</i>	pixel at the given position

5.23.2.2 read() [2/4]

```
access_type read (
    array< U, DIM > pos,
    sampler_mode sampler = address_none | filter_nearest ) const [inline]
```

read from image slice with sampler

Parameters

<i>pos</i>	position, may be an array of type gpu_int, gpu_uint, or gpu_float
<i>sampler</i>	mode.
<i>return</i>	pixel

5.23.2.3 read() [3/4]

```
access_type read (
    POS pos ) const [inline]
```

read pixel.

Parameters

<i>pos</i>	pixel position. May be any container that provides the [] operator
<i>return</i>	pixel at the given position

5.23.2.4 read() [4/4]

```
access_type read (
    POS pos,
    sampler_mode sampler ) const [inline]
```

read from image with sampler

Parameters

<i>pos</i>	position, may be any container that provides the [] operator, and that holds values of type gpu_int, gpu_uint, or gpu_float
<i>sampler</i>	mode.
<i>return</i>	pixel

5.23.2.5 write() [1/2]

```
void write (
    array< size_type, DIM > pos,
    access_type colors ) [inline]
```

write to image

Parameters

<i>pos</i>	pixel position
<i>colors</i>	pixel color to write

5.23.2.6 write() [2/2]

```
void write (
    POS pos,
    access_type colors ) [inline]
```

write to image

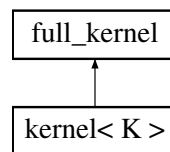
Parameters

<i>pos</i>	pixel position
<i>colors</i>	pixel color to write. May be any container that provides the [] operator

5.24 kernel< K > Struct Template Reference

kernel class to derive gpu kernels from

Inheritance diagram for kernel< K >:



Public Member Functions

- `template<class... A>`
`auto operator() (goopax_device device, A &&... a)`
Operator (). This starts the gpu kernel on the device. The parameters must match the arguments specified in the function 'program()', except that cpu types are passed, where gpu types are specified in the function 'program()'. The device may be specified as a first argument. This is optional except if the device cannot be deduced from the other arguments. All buffers must reside on the same device, which is the device on which the kernel is executed.
- `template<class... A>`
`auto operator() (A &&... a)`
- `kernel ()`
- `kernel (kernel &&)=default`
move semantics
- `kernel & operator= (kernel &&b)`

5.24.1 Detailed Description

```

template<class K>
struct goopax::impl::ns_param_kernel::kernel< K >
  
```

kernel class to derive gpu kernels from

Parameters

<i>K</i>	derived kernel class. If The C++ language standard is < 14, any return type of the kernel must be specified with the RETURN_TYPE template argument. All gpu kernels must be derived from this class, where the new class must be passed as template argument, like so: <code>struct my_kernel : public kernel<my_kernel> { void program(...) {...} };</code>
----------	--

5.24.2 Constructor & Destructor Documentation

5.24.2.1 kernel()

```
kernel ( ) [inline]
```

constructor

5.24.3 Member Function Documentation

5.24.3.1 operator>()

```
auto operator() (
    A &&... a ) [inline]
```

Operator (). This starts the gpu kernel on the device. The parameters must match the arguments specified in the function 'program()', except that cpu types are passed, where gpu types are specified in the function 'program()'. The device is deduced from the buffers that are passed as arguments.

5.24.3.2 operator=()

```
kernel& operator= (
    kernel< K > && b ) [inline]
```

move semantics

5.25 local_scope_mem< T, SCOPE > Class Template Reference

Inherits lds_type< T, goopax::impl::Tuint, SCOPE >.

Public Member Functions

- `cpu_size_type size () const`
- `iterator begin ()`
- `const_iterator begin () const`
- `iterator end ()`
- `const_iterator end () const`
- `template<typename RESOURCE_SIZE_TYPE , bool is_const>`
`void copy (const resource< T, RESOURCE_SIZE_TYPE, is_const > &r, size_type count, typename resource< T, RESOURCE_SIZE_TYPE, is_const >::size_type start_dest, size_type start_src)`
- `template<typename RESOURCE_SIZE_TYPE , bool is_const>`
`void copy (const resource< T, RESOURCE_SIZE_TYPE, is_const > &r)`
- `template<memory::scope BSCOPE>`
`void copy (const local_scope_mem< T, BSCOPE > &b, size_t count, typename local_scope_mem< T, BSCOPE >::size_type start_dest, size_type start_src)`
- `template<memory::scope BSCOPE>`
`void copy (const local_scope_mem< T, BSCOPE > &b)`
- `void fill (value_type value)`
- `local_scope_mem (local_scope_mem &&)=default`
move semantics
- `local_scope_mem & operator= (local_scope_mem &&)=default`
move semantics
- `local_scope_mem (typename gettype< size_type >::cpu size)`

5.25.1 Detailed Description

```
template<typename T, memory::scope SCOPE>
class goopax::impl::local_scope_mem< T, SCOPE >
```

This class is used for local and private memory.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 local_scope_mem()

```
local_scope_mem (
    typename gettype< size_type >::cpu size ) [inline]
```

constructor.

Parameters

<i>size</i>	size of the local/private memory
-------------	----------------------------------

5.25.3 Member Function Documentation

5.25.3.1 begin() [1/2]

```
iterator begin ( ) [inline]
```

returns `begin()` pointer

5.25.3.2 begin() [2/2]

```
const_iterator begin ( ) const [inline]
```

returns `begin()` pointer

5.25.3.3 copy() [1/4]

```
void copy (
    const resource< T, RESOURCE_SIZE_TYPE, is_const > & r,
    size_type count,
    typename resource< T, RESOURCE_SIZE_TYPE, is_const >::size_type start_dest,
    size_type start_src ) [inline]
```

copy data from resource. For local memory, the copy operation is done in parallel in the workgroup. For private memory, each thread performs its own copy operation.

Parameters

<i>r</i>	resource to copy data from
<i>count</i>	size number of values to copy
<i>start_dest</i>	offset in local/private memory
<i>start_src</i>	offset in the source resource

5.25.3.4 copy() [2/4]

```
void copy (
    const resource< T, RESOURCE_SIZE_TYPE, is_const > & r ) [inline]
```

copy data from resource. For local memory, the copy operation is done in parallel in the workgroup. For private memory, each thread performs its own copy operation.

Parameters

<i>r</i>	resource to copy data from
----------	----------------------------

5.25.3.5 `copy()` [3/4]

```
void copy (
    const local_scope_mem< T, BSCOPE > & b,
    size_t count,
    typename local_scope_mem< T, BSCOPE >::size_type start_dest,
    size_type start_src ) [inline]
```

copy data from another local/private memory. For local memory, the copy operation is done in parallel in the workgroup. For private memory, each thread performs its own copy operation.

Parameters

<i>b</i>	local/private memory to copy data from
<i>count</i>	number of elements to copy
<i>start_dest</i>	offset in destination memory
<i>start_src</i>	offset in source memory

5.25.3.6 `copy()` [4/4]

```
void copy (
    const local_scope_mem< T, BSCOPE > & b ) [inline]
```

copy data from another local/private memory. For local memory, the copy operation is done in parallel in the workgroup. For private memory, each thread performs its own copy operation.

Parameters

<i>b</i>	local/private memory to copy data from
----------	--

5.25.3.7 `end()` [1/2]

```
iterator end ( ) [inline]
```

returns `end()` iterator

5.25.3.8 `end()` [2/2]

```
const_iterator end ( ) const [inline]
```

returns `end()` iterator

5.25.3.9 fill()

```
void fill (
    value_type value ) [inline]
```

fill local/private memory. For local memory, the fill operation is done in parallel over the workgroup.

Parameters

<i>value</i>	value to fill memory with
--------------	---------------------------

5.25.3.10 size()

```
cpu_size_type size ( ) const [inline]
```

size of local/private memory

5.26 numeric_limits< goopax::gpu_type< T > > Struct Template Reference

numeric_limits overload for GPU types

Inherits numeric_limits< T >.

5.26.1 Detailed Description

```
template<typename T>
struct std::numeric_limits< goopax::gpu_type< T > >
```

numeric_limits overload for GPU types

5.27 resource< T, SIZE_TYPE, is_const > Class Template Reference

Inherits uav_type< T, gettype< SIZE_TYPE >::cpu, memory::none >.

Public Member Functions

- `template<memory::scope BSCOPE, typename EIF = void, typename = typename enable_if<!is_const, EIF>::type>`
`void copy (const local_scope_mem< T, BSCOPE > &src, typename local_scope_mem< T, BSCOPE >::size_type count, typename local_scope_mem< T, BSCOPE >::size_type start_dest, size_type start_src=0)`
- `resource (resource &&)=default`
move semantics
- `resource & operator= (resource &&)=default`
move semantics

iterator access

- `iterator begin ()`
- `const_iterator begin () const`
- `iterator end ()`
- `const_iterator end () const`

constructors

Resources normally need not be constructed by the user. Instead, they are passed as references to the gpu kernel. Resources can, however, be instantiated to be linked to a specific buffer

- `resource ()`
- `resource (buffer_type &a)`
- `template<typename EIF = void, typename = typename enable_if<!is_const, EIF>::type>`
`resource (const buffer_type &a)`

5.27.1 Detailed Description

```
template<typename T, typename SIZE_TYPE = Tuint, bool is_const = false>
class goopax::impl::resource< T, SIZE_TYPE, is_const >
```

Resource type, used to access buffers from device code. Resources are usually provided as arguments for gpu kernels. The template parameters T and SIZE_TYPE must match the corresponding template parameters of the buffer.

Template Parameters

<i>T</i>	value type. This can either be a CPU intrinsic type, or a goopax struct type.
<i>SIZE_TYPE</i>	This can be a 32 bit or 64 bit integer type.
<i>is_const</i>	only read access is allowed. This has the same effect as using a const reference of the resource as kernel parameter.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 resource() [1/3]

```
resource ( ) [inline]
```


default constructor. The resulting resource cannot be used. It has to be replaced by a resource that is linked to a buffer before it can be used.

5.27.2.2 resource() [2/3]

```
resource (
    buffer_type & a ) [inline]
```

constructor. The resource is linked to the provided buffer

Parameters

<code>a</code>	buffer to link resource to. The buffer must exist and be properly initialized before the kernel is first called.
----------------	--

5.27.2.3 resource() [3/3]

```
resource (
    const buffer_type & a ) [inline]
```

constructor. The resource is linked to the provided const buffer This constructor is only available if `is_const=true`.

Parameters

<code>a</code>	buffer to link resource to. The buffer must exist and be properly initialized before the kernel is first called.
----------------	--

5.27.3 Member Function Documentation

5.27.3.1 begin() [1/2]

```
iterator begin ( ) [inline]
```

returns `begin()` pointer

5.27.3.2 begin() [2/2]

```
const_iterator begin ( ) const [inline]
```

returns `begin()` pointer

5.27.3.3 copy()

```
void copy (
    const local_scope_mem< T, BSCOPE > & src,
    typename local_scope_mem< T, BSCOPE >::size_type count,
    typename local_scope_mem< T, BSCOPE >::size_type start_dest,
    size_type start_src = 0 ) [inline]
```

copy data from local memory or private memory to the resource (i.e. to global memory) For local memory, the copy operation is parallelized over the workgroup. For private memory, the copy operation is performed for the individual threads.

Parameters

<i>src</i>	source
<i>count</i>	number of values to copy
<i>start_dest</i>	offset in resource
<i>start_src</i>	offset in source memory

5.27.3.4 end() [1/2]

```
iterator end ( ) [inline]
```

pointer to the end of the resource

5.27.3.5 end() [2/2]

```
const_iterator end ( ) const [inline]
```

pointer to the end of the resource

5.28 sresource< STR, BASE > Struct Template Reference

Public Member Functions

- reference [operator\[\]](#) (const size_type &pos)
element access
- const_reference [operator\[\]](#) (const size_type &pos) const
element access

5.28.1 Detailed Description

```
template<class STR, class BASE>
struct goopax::impl::gpu_struct::sresource< STR, BASE >
```

base for resource class.

5.29 WELL512 Class Reference

Public Member Functions

- void `seed` (`goopax_device` device, const vector< Tuint32_t > &seed)

5.29.1 Detailed Description

Random number generator, based on the [WELL512](#) algorithm. An object of this class must be instantiated to be used from kernels.

5.29.2 Member Function Documentation

5.29.2.1 `seed()`

```
void seed (
    goopax_device device,
    const vector< Tuint32_t > & seed ) [inline]
```

Set random seed. All threads will get an additional offset by goopax, so that they all produce different random numbers.

Parameters

<i>device</i>	device for which to set the random seed
<i>seed</i>	seed data, may be any size, maximum 16.

5.30 WELL512lib Class Reference

Public Member Functions

- template<typename T >
vector< typename `gettype`< T >::gpu > `gen_vec` ()
- [WELL512lib](#) ([WELL512](#) &well)

5.30.1 Detailed Description

Random number generator, used in device code This object must be linked to an [WELL512](#) object in host code

5.30.2 Constructor & Destructor Documentation

5.30.2.1 WELL512lib()

```
WELL512lib (
    WELL512 & well ) [inline]
```

Constructor. Must be linked to an existing [WELL512](#) object.

Parameters

<i>well</i>	existing WELL512 object.
-------------	--

5.30.3 Member Function Documentation

5.30.3.1 gen_vec()

```
vector<typename gettype<T>::gpu> gen_vec ( ) [inline]
```

Generate uniformly distributed random numbers of type T. With each call, a vector of values is created. For optimal performance, all values in the vector should be used. Floating point values are normalized in the range [0,1). Integer values are equally distributed over all possible values.

Index

- abs
 - cpu and gpu functions, 27
- acospi
 - cpu and gpu functions, 27
- array_size
 - goopax::impl::image_array_buffer, 96
 - goopax::impl::image_array_buffer_map, 101
- asinpi
 - cpu and gpu functions, 27, 28
- atanpi
 - cpu and gpu functions, 28
- atomic operations, 7
 - atomic_add, 8
 - atomic_and, 8
 - atomic_cmpxchg, 8–11
 - atomic_load, 12, 13
 - atomic_max, 13
 - atomic_min, 13
 - atomic_or, 14
 - atomic_sub, 14, 16, 17
 - atomic_xchg, 17, 18
 - atomic_xor, 18
- atomic_add
 - atomic operations, 8
- atomic_and
 - atomic operations, 8
- atomic_cmpxchg
 - atomic operations, 8–11
- atomic_load
 - atomic operations, 12, 13
- atomic_max
 - atomic operations, 13
- atomic_min
 - atomic operations, 13
- atomic_or
 - atomic operations, 14
- atomic_sub
 - atomic operations, 14, 16, 17
- atomic_xchg
 - atomic operations, 17, 18
- atomic_xor
 - atomic operations, 18
- BUFFER_FLAGS
 - memory resources, 20
- ballot
 - Thread_communication, 60
- begin
 - goopax::impl::buffer_map, 79
 - goopax::impl::local_scope_mem, 123
 - goopax::impl::resource, 127
- buffer
 - goopax::impl::buffer, 71
- buffer< T, SIZE_TYPE >, 69
- buffer_map
 - goopax::impl::buffer_map, 77, 78
- buffer_map< T, is_const >, 76
- CUDA interoperability, 33
 - create_from_cuda, 33
 - get_cuda_device, 33
- clamp
 - cpu and gpu functions, 28
- clz
 - cpu and gpu functions, 29
- cond
 - Operators, 48, 49
- copy
 - goopax::impl::buffer, 72
 - goopax::impl::image_array_buffer, 96
 - goopax::impl::image_array_buffer::image_slice, 114, 115
 - goopax::impl::image_buffer, 105, 106
 - goopax::impl::local_scope_mem, 123, 124
 - goopax::impl::resource, 127
- copy_from_host
 - goopax::impl::buffer, 72, 73
 - goopax::impl::image_array_buffer, 96
 - goopax::impl::image_array_buffer::image_slice, 115
 - goopax::impl::image_buffer, 106
- copy_to_host
 - goopax::impl::buffer, 73
 - goopax::impl::image_array_buffer, 97
 - goopax::impl::image_array_buffer::image_slice, 115, 116
 - goopax::impl::image_buffer, 107
- cospi
 - cpu and gpu functions, 29
- cpu and gpu functions, 26
 - abs, 27
 - acospi, 27
 - asinpi, 27, 28
 - atanpi, 28
 - clamp, 28
 - clz, 29
 - cospi, 29
 - max, 29, 30
 - min, 30
 - popcount, 30, 31

- rotl, 31
- rotr, 31
- sinpi, 31, 32
- tanpi, 32
- create_from_cl
 - OpenCL interoperability, 22, 24
- create_from_cuda
 - CUDA interoperability, 33
- create_from_gl
 - OpenGL interoperability, 42, 43
- data
 - goopax::impl::buffer_map, 79
- debugging, 46
 - gpu_assert, 46
- default_device
 - Device, 37
- Device, 37
 - default_device, 37
 - devices, 38
 - envmode, 37
 - get_current_build_device, 38
 - init_params, 38
 - operator |, 40
- devices
 - Device, 38
- dimensions
 - goopax::impl::image_array_buffer::image_slice, 116
 - goopax::impl::image_array_buffer_map, 101
 - goopax::impl::image_buffer_map, 110
- EX::goopax_exception, 85
- end
 - goopax::impl::buffer_map, 79
 - goopax::impl::local_scope_mem, 124
 - goopax::impl::resource, 128
- envmode
 - Device, 37
- fill
 - goopax::impl::buffer, 74
 - goopax::impl::image_array_buffer, 98
 - goopax::impl::image_buffer, 107
 - goopax::impl::local_scope_mem, 124
- flush_gl_interop
 - OpenGL interoperability, 43
- full_kernel, 80
- Gather return values, 41
- gather_add< T >, 81
- gather_max< T >, 81
- gather_min< T >, 82
- gen_vec
 - goopax::impl::WELL512lib, 130
- get
 - goopax::impl::goopax_future, 86
- get_cl_context
 - OpenCL interoperability, 24
- get_cl_device
 - OpenCL interoperability, 24
- get_cl_device_extensions
 - OpenCL interoperability, 24
- get_cl_mem
 - OpenCL interoperability, 24
- get_cl_platform
 - OpenCL interoperability, 25
- get_cl_platform_extensions
 - OpenCL interoperability, 25
- get_cl_queue
 - OpenCL interoperability, 25
- get_cuda_device
 - CUDA interoperability, 33
- get_current_build_device
 - Device, 38
- get_device
 - goopax::impl::buffer, 74
- get_device_from_cl_queue
 - OpenCL interoperability, 25
- gettype< T, EIF >, 82
- global_id
 - Thread numbers, 34
- global_size
 - Thread numbers, 34
- goopax struct types, 59
 - output_goopax_struct, 59
- goopax::impl::WELL512
 - seed, 129
- goopax::impl::WELL512lib
 - gen_vec, 130
 - WELL512lib, 129
- goopax::impl::buffer
 - buffer, 71
 - copy, 72
 - copy_from_host, 72, 73
 - copy_to_host, 73
 - fill, 74
 - get_device, 74
 - max, 74
 - min, 75
 - operator<<, 76
 - operator=, 75
 - reinterpret, 76
 - size, 75
 - sum, 75
 - to_vector, 76
- goopax::impl::buffer_map
 - begin, 79
 - buffer_map, 77, 78
 - data, 79
 - end, 79
 - operator[], 79, 80
- goopax::impl::gather_add
 - init, 81
 - op, 81
- goopax::impl::goopax_device
 - goopax_device, 84

- operator=, [84](#)
- goopax::impl::goopax_future
 - get, [86](#)
- goopax::impl::gpu_ostream
 - gpu_ostream, [88](#)
- goopax::impl::gpu_type
 - gpu_break, [92](#)
 - gpu_continue, [92](#)
 - gpu_type, [91](#), [92](#)
 - operator<=, [93](#)
 - operator>, [94](#)
 - operator+, [92](#)
 - operator++, [93](#)
 - operator-, [93](#)
 - operator--, [93](#)
 - operator=, [94](#)
- goopax::impl::image_array_buffer
 - array_size, [96](#)
 - copy, [96](#)
 - copy_from_host, [96](#)
 - copy_to_host, [97](#)
 - fill, [98](#)
 - image_array_buffer, [95](#)
 - operator[], [98](#)
 - to_vector, [98](#)
- goopax::impl::image_array_buffer::image_slice
 - copy, [114](#), [115](#)
 - copy_from_host, [115](#)
 - copy_to_host, [115](#), [116](#)
 - dimensions, [116](#)
 - to_vector, [116](#)
- goopax::impl::image_array_buffer_map
 - array_size, [101](#)
 - dimensions, [101](#)
 - image_array_buffer_map, [99](#), [100](#)
 - operator[], [101](#)
- goopax::impl::image_array_resource
 - image_array_resource, [102](#)
 - operator[], [103](#)
- goopax::impl::image_array_resource::image_slice
 - read, [117](#), [119](#)
 - write, [119](#)
- goopax::impl::image_buffer
 - copy, [105](#), [106](#)
 - copy_from_host, [106](#)
 - copy_to_host, [107](#)
 - fill, [107](#)
 - image_buffer, [105](#)
 - to_vector, [108](#)
- goopax::impl::image_buffer_map
 - dimensions, [110](#)
 - image_buffer_map, [108](#), [109](#)
 - operator[], [110](#)
- goopax::impl::image_resource
 - image_resource, [111](#)
 - read, [112](#), [113](#)
 - write, [113](#)
- goopax::impl::local_scope_mem
 - begin, [123](#)
 - copy, [123](#), [124](#)
 - end, [124](#)
 - fill, [124](#)
 - local_scope_mem, [122](#)
 - size, [125](#)
- goopax::impl::ns_param_kernel::kernel
 - kernel, [121](#)
 - operator(), [121](#)
 - operator=, [121](#)
- goopax::impl::resource
 - begin, [127](#)
 - copy, [127](#)
 - end, [128](#)
 - resource, [126](#), [127](#)
- goopax_device, [82](#)
 - goopax::impl::goopax_device, [84](#)
- goopax_future< T >, [85](#)
- goopax_future< void >, [86](#)
- goopax_future_info, [87](#)
- goopax_struct_type< std::array< T, N > >, [87](#)
- gpu_assert
 - debugging, [46](#)
- gpu_break
 - goopax::impl::gpu_type, [92](#)
- gpu_continue
 - goopax::impl::gpu_type, [92](#)
- gpu_else
 - Loops, [53](#)
- gpu_elseif
 - Loops, [54](#)
- gpu_for_global
 - Loops, [54](#)
- gpu_for_group
 - Loops, [54](#)
- gpu_for_local
 - Loops, [54](#)
- gpu_if
 - Loops, [54](#)
- gpu_ostream, [87](#)
 - goopax::impl::gpu_ostream, [88](#)
- gpu_type
 - goopax::impl::gpu_type, [91](#), [92](#)
- gpu_type< T, scope >, [89](#)
- gpu_while
 - Loops, [54](#)
- group_id
 - Thread numbers, [35](#)
- have_cl_device_extension
 - OpenCL interoperability, [25](#)
- have_cl_platform_extension
 - OpenCL interoperability, [25](#)
- IMAGE_FLAGS
 - Image, [51](#)
- Image, [51](#)
 - IMAGE_FLAGS, [51](#)
- image_array_buffer

- goopax::impl::image_array_buffer, 95
- image_array_buffer< DIM, T, normalized >, 94
- image_array_buffer< DIM, T, normalized >::image_↔
slice< is_const >, 114
- image_array_buffer_map
 - goopax::impl::image_array_buffer_map, 99, 100
- image_array_buffer_map< DIM, T, is_const >, 99
- image_array_resource
 - goopax::impl::image_array_resource, 102
- image_array_resource< DIM, T, normalized >, 102
- image_array_resource< DIM, T, normalized >::image_↔
_slice< RESREF >, 116
- image_buffer
 - goopax::impl::image_buffer, 105
- image_buffer< DIM, T, normalized >, 103
- image_buffer_map
 - goopax::impl::image_buffer_map, 108, 109
- image_buffer_map< DIM, T, is_const >, 108
- image_resource
 - goopax::impl::image_resource, 111
- image_resource< DIM, T, normalized >, 110
- init
 - goopax::impl::gather_add, 81
- init_params
 - Device, 38
- Kernel, 52
- kernel
 - goopax::impl::ns_param_kernel::kernel, 121
- kernel< K >, 120
- local_id
 - Thread numbers, 35
- local_mem
 - memory resources, 20
- local_scope_mem
 - goopax::impl::local_scope_mem, 122
- local_scope_mem< T, SCOPE >, 121
- local_size
 - Thread numbers, 35
- Loops, 53
 - gpu_else, 53
 - gpu_elseif, 54
 - gpu_for_global, 54
 - gpu_for_group, 54
 - gpu_for_local, 54
 - gpu_if, 54
 - gpu_while, 54
- Math, 44
 - pow, 44, 45
- max
 - cpu and gpu functions, 29, 30
 - goopax::impl::buffer, 74
- max_local_mem
 - Thread numbers, 35
- memory resources, 20
 - BUFFER_FLAGS, 20
 - local_mem, 20
 - private_mem, 20
- min
 - cpu and gpu functions, 30
 - goopax::impl::buffer, 75
- num_groups
 - Thread numbers, 35
- num_subthreads
 - Thread numbers, 35
- numeric_limits< goopax::gpu_type< T > >, 125
- op
 - goopax::impl::gather_add, 81
- OpenCL interoperability, 22
 - create_from_cl, 22, 24
 - get_cl_context, 24
 - get_cl_device, 24
 - get_cl_device_extensions, 24
 - get_cl_mem, 24
 - get_cl_platform, 25
 - get_cl_platform_extensions, 25
 - get_cl_queue, 25
 - get_device_from_cl_queue, 25
 - have_cl_device_extension, 25
 - have_cl_platform_extension, 25
- OpenGL interoperability, 42
 - create_from_gl, 42, 43
 - flush_gl_interop, 43
- operator<<
 - goopax::impl::buffer, 76
- operator<=
 - goopax::impl::gpu_type, 93
- operator>
 - goopax::impl::gpu_type, 94
- operator()
 - goopax::impl::ns_param_kernel::kernel, 121
- operator+
 - goopax::impl::gpu_type, 92
- operator++
 - goopax::impl::gpu_type, 93
- operator-
 - goopax::impl::gpu_type, 93
- operator--
 - goopax::impl::gpu_type, 93
- operator=
 - goopax::impl::buffer, 75
 - goopax::impl::goopax_device, 84
 - goopax::impl::gpu_type, 94
 - goopax::impl::ns_param_kernel::kernel, 121
- operator[]
 - goopax::impl::buffer_map, 79, 80
 - goopax::impl::image_array_buffer, 98
 - goopax::impl::image_array_buffer_map, 101
 - goopax::impl::image_array_resource, 103
 - goopax::impl::image_buffer_map, 110
- operator |
 - Device, 40
- Operators, 48
 - cond, 48, 49

- operators, 55
- output_goopax_struct
 - goopax struct types, 59
- popcount
 - cpu and gpu functions, 30, 31
- pow
 - Math, 44, 45
- private_mem
 - memory resources, 20
- random numbers, 58
- read
 - goopax::impl::image_array_resource::image_slice, 117, 119
 - goopax::impl::image_resource, 112, 113
- Reinterpret, 56
 - reinterpret, 56
- reinterpret
 - goopax::impl::buffer, 76
 - Reinterpret, 56
- resource
 - goopax::impl::resource, 126, 127
- resource< T, SIZE_TYPE, is_const >, 125
- rotl
 - cpu and gpu functions, 31
- rotr
 - cpu and gpu functions, 31
- seed
 - goopax::impl::WELL512, 129
- shuffle
 - Thread_communication, 61
- sinpi
 - cpu and gpu functions, 31, 32
- size
 - goopax::impl::buffer, 75
 - goopax::impl::local_scope_mem, 125
- sresource< STR, BASE >, 128
- sum
 - goopax::impl::buffer, 75
- tanpi
 - cpu and gpu functions, 32
- Thread numbers, 34
 - global_id, 34
 - global_size, 34
 - group_id, 35
 - local_id, 35
 - local_size, 35
 - max_local_mem, 35
 - num_groups, 35
 - num_subthreads, 35
 - thread_id_in_warp, 35
 - warp_id, 36
 - warp_id_in_group, 36
 - warp_size, 36
- Thread_communication, 60
 - ballot, 60
 - shuffle, 61
 - work_group_all, 61
 - work_group_any, 62
 - work_group_broadcast, 62
 - work_group_reduce, 62
 - work_group_reduce_add, 63
 - work_group_reduce_max, 64, 65
 - work_group_reduce_min, 64, 65
 - work_group_scan_exclusive_add, 65
 - work_group_scan_exclusive_max, 65
 - work_group_scan_exclusive_min, 66
 - work_group_scan_inclusive_add, 66
 - work_group_scan_inclusive_max, 66
 - work_group_scan_inclusive_min, 67
- thread_id_in_warp
 - Thread numbers, 35
- to_vector
 - goopax::impl::buffer, 76
 - goopax::impl::image_array_buffer, 98
 - goopax::impl::image_array_buffer::image_slice, 116
 - goopax::impl::image_buffer, 108
- Types, 47
 - WELL512, 129
 - WELL512lib, 129
 - goopax::impl::WELL512lib, 129
- warp_id
 - Thread numbers, 36
- warp_id_in_group
 - Thread numbers, 36
- warp_size
 - Thread numbers, 36
- work_group_all
 - Thread_communication, 61
- work_group_any
 - Thread_communication, 62
- work_group_broadcast
 - Thread_communication, 62
- work_group_reduce
 - Thread_communication, 62
- work_group_reduce_add
 - Thread_communication, 63
- work_group_reduce_max
 - Thread_communication, 64
- work_group_reduce_min
 - Thread_communication, 64, 65
- work_group_scan_exclusive_add
 - Thread_communication, 65
- work_group_scan_exclusive_max
 - Thread_communication, 65
- work_group_scan_exclusive_min
 - Thread_communication, 66
- work_group_scan_inclusive_add
 - Thread_communication, 66
- work_group_scan_inclusive_max
 - Thread_communication, 66
- work_group_scan_inclusive_min
 - Thread_communication, 67

write

goopax::impl::image_array_resource::image_slice,
[119](#)

goopax::impl::image_resource, [113](#)