OpenCL™ (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other devices. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems, and handheld devices. Specification documents and online reference are available at www.khronos.org/opencl.

▢: Content relating to optional features in OpenCL 3.0

[API n.n.n] OpenCL 3.0 API specification
[C n.n.n] OpenCL 3.0 C Language specification
[Ext n.n.n] OpenCL 3.0 Extension specification

OpenCL API Reference

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices.

Querying platform info & devices [API 4.1]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*, cl_platform_id **platforms*, cl_uint **num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*, cl_platform_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PLATFORM_*X* where *X* may be:
EXTENSIONS, EXTENSIONS_WITH_VERSION,
▢ HOST_TIMER_RESOLUTION, NAME,
NUMERIC_VERSION, PROFILE, VENDOR, VERSION

cl_int **clGetDeviceIDs** (cl_platform_id *platform*, cl_device_type *device_type*, cl_uint *num_entries*, cl_device_id **devices*, cl_uint **num_devices*)

device_type:
CL_DEVICE_TYPE_{ACCELERATOR, ALL, CPU},
CL_DEVICE_TYPE_{CUSTOM, DEFAULT, GPU}

cl_int **clGetDeviceInfo** (cl_device_id *device*, cl_device_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_DRIVER_VERSION or
CL_DEVICE_*X* where *X* may be:
ADDRESS_BITS, CL_DEVICE_AVAILABLE,
▢ ATOMIC_FENCE_CAPABILITIES,
▢ ATOMIC_MEMORY_CAPABILITIES,
BUILT_IN_KERNELS,
COMPILER_AVAILABLE,
▢ DEVICE_ENQUEUE_SUPPORT,
DOUBLE_FP_CONFIG,
ENDIAN_LITTLE,
EXTENSIONS, EXTENSIONS_WITH_VERSION,
ERROR_CORRECTION_SUPPORT,
EXECUTION_CAPABILITIES,
▢ GENERIC_ADDRESS_SPACE_SUPPORT,
GLOBAL_MEM_CACHE_SIZE,
GLOBAL_MEM_CACHE_TYPE,
GLOBAL_MEM_CACHELINE_SIZE,
GLOBAL_MEM_SIZE,
▢ GLOBAL_VARIABLE_PREFERRED_TOTAL_SIZE,
▢ IL_VERSION,
▢ ILS_WITH_VERSION,
IMAGE_MAX_ARRAY_SIZE,
IMAGE_MAX_BUFFER_SIZE,
IMAGE_SUPPORT,
IMAGE2D_MAX_HEIGHT, IMAGE2D_MAX_WIDTH,
IMAGE3D_MAX_DEPTH, IMAGE3D_MAX_HEIGHT,
IMAGE3D_MAX_WIDTH,
▢ IMAGE_BASE_ADDRESS_ALIGNMENT,
▢ IMAGE_PITCH_ALIGNMENT,
LINKER_AVAILABLE,
LOCAL_MEM_SIZE, LOCAL_MEM_TYPE,
MAX_CLOCK_FREQUENCY,
▢ MAX_PIPE_ARGS,
MAX_COMPUTE_UNITS,
MAX_SAMPLERS,
MAX_CONSTANT_ARGS,
MAX_CONSTANT_BUFFER_SIZE,
▢ MAX_GLOBAL_VARIABLE_SIZE,
MAX_MEM_ALLOC_SIZE,
MAX_PARAMETER_SIZE,
▢ MAX_NUM_SUB_GROUPS,
▢ MAX_ON_DEVICE_EVENTS,
▢ MAX_ON_DEVICE_QUEUES,
MAX_READ_IMAGE_ARGS,
▢ MAX_READ_WRITE_IMAGE_ARGS,
MAX_WRITE_IMAGE_ARGS,
▢ MAX_SUB_GROUPS,
MAX_WORK_GROUP_SIZE,
MAX_WORK_ITEM_DIMENSIONS,
MAX_WORK_ITEM_SIZES,
MEM_BASE_ADDR_ALIGN,
NAME,
NATIVE_VECTOR_WIDTH_{CHAR, INT, DOUBLE, HALF},
NATIVE_VECTOR_WIDTH_{LONG, SHORT, FLOAT},
▢ NON_UNIFORM_WORK_GROUP_SUPPORT,
OPENCL_C_VERSION, OPENCL_C_ALL_VERSIONS,
OPENCL_C_FEATURES,
PARENT_DEVICE,
PARTITION_AFFINITY_DOMAIN,
PARTITION_MAX_SUB_DEVICES,
PARTITION_PROPERTIES,
PARTITION_TYPE,
▢ PIPE_MAX_ACTIVE_RESERVATIONS,
▢ PIPE_MAX_PACKET_SIZE,
▢ PIPE_SUPPORT,
PLATFORM,
PRINTF_BUFFER_SIZE,
PREFERRED_GLOBAL_ATOMIC_ALIGNMENT,
PREFERRED_LOCAL_ATOMIC_ALIGNMENT,
PREFERRED_PLATFORM_ATOMIC_ALIGNMENT,
PREFERRED_VECTOR_WIDTH_{CHAR, INT, DOUBLE, HALF},
PREFERRED_VECTOR_WIDTH_{LONG, SHORT, FLOAT},
PREFERRED_INTEROP_USER_SYNC,
PROFILE,
PROFILING_TIMER_RESOLUTION,
▢ QUEUE_ON_DEVICE_PROPERTIES,
QUEUE_ON_HOST_PROPERTIES,
▢ QUEUE_ON_DEVICE_MAX_SIZE,
▢ QUEUE_ON_DEVICE_PREFERRED_SIZE,
REFERENCE_COUNT,
SINGLE_FP_CONFIG,
▢ SUB_GROUP_INDEPENDENT_FORWARD_PROGRESS,
▢ SVM_CAPABILITIES,
TYPE,
VENDOR, VENDOR_ID,
VERSION,
▢ WORK_GROUP_COLLECTIVE_FUNCTIONS_SUPPORT

▢ cl_int **clGetDeviceAndHostTimer** (cl_device_id *device*, cl_ulong **device_timestamp*, cl_ulong **host_timestamp*)

▢ cl_int **clGetHostTimer** (cl_device_id *device*, cl_ulong **host_timestamp*)

Partitioning a device [API 4.3]

cl_int **clCreateSubDevices** (cl_device_id *in_device*, const cl_device_partition_property **properties*, cl_uint *num_devices*, cl_device_id **out_devices*, cl_uint **num_devices_ret*)

properties: CL_DEVICE_PARTITION_EQUALLY,
CL_DEVICE_PARTITION_BY_COUNTS,
CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN

cl_int **clRetainDevice** (cl_device_id *device*)

cl_int **clReleaseDevice** (cl_device_id *device*)

Contexts [API 4.4]

cl_context **clCreateContext** (
const cl_context_properties **properties*,
cl_uint *num_devices*, const cl_device_id **devices*,
void (CL_CALLBACK**pfn_notify*)
(const char **errinfo*, const void **private_info*, size_t *cb*, void **user_data*),
void **user_data*, cl_int **errcode_ret*)

properties:
NULL or CL_CONTEXT_PLATFORM,
CL_CONTEXT_INTEROP_USER_SYNC

cl_context **clCreateContextFromType** (
const cl_context_properties **properties*,
cl_device_type *device_type*,
void (CL_CALLBACK **pfn_notify*)
(const char **errinfo*, const void **private_info*, size_t *cb*, void **user_data*),
void **user_data*, cl_int **errcode_ret*)

properties: See **clCreateContext**
device_type: See **clGetDeviceIDs**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

cl_int **clGetContextInfo** (cl_context *context*, cl_context_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name:
CL_CONTEXT_*X* where *X* may be REFERENCE_COUNT, DEVICES, NUM_DEVICES, PROPERTIES

Get CL extension function pointers [Ext 1.3]

void* **clGetExtensionFunctionAddressForPlatform** (cl_platform_id *platform*, const char **funcname*)

The OpenCL Runtime

API calls that manage OpenCL objects such as command-queues, memory objects, program objects, kernel objects for __kernel functions in a program and calls that allow you to enqueue commands to a command-queue such as executing a kernel, reading, or writing a memory object.

Command queues [API 5.1]

cl_command_queue
clCreateCommandQueueWithProperties (
cl_context *context*, cl_device_id *device*,
const cl_command_queue_properties **properties*,
cl_int **errcode_ret*)

**properties:* NULL or a pointer to a zero-terminated list of properties and their values:
▢ CL_QUEUE_SIZE,
CL_QUEUE_PROPERTIES (bitfield which may be set to an OR of CL_QUEUE_* where * may be: OUT_OF_ORDER_EXEC_MODE_ENABLE, PROFILING_ENABLE, ▢ ON_DEVICE[_DEFAULT]),

▢ cl_int **clSetDefaultDeviceCommandQueue** (
cl_context *context*, cl_device_id *device*,
cl_command_queue *command_queue*)

cl_int **clRetainCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (
cl_command_queue *command_queue*,
cl_command_queue_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)

param_name:
CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE,
▢ CL_QUEUE_DEVICE_DEFAULT, ▢ CL_QUEUE_SIZE,
CL_QUEUE_REFERENCE_COUNT,
CL_QUEUE_PROPERTIES

Buffer Objects [API 5.2]

Elements of buffer objects are stored sequentially and accessed using a pointer by a kernel executing on a device.

Create buffer objects

cl_mem **clCreateBuffer** (
cl_context *context*, cl_mem_flags *flags*, size_t *size*,
void *host_ptr, cl_int *errcode_ret)

flags: CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_HOST_NO_ACCESS, CL_MEM_HOST_{READ, WRITE}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

cl_mem **clCreateBufferWithProperties** (
cl_context *context*, const cl_mem_properties *properties, cl_mem_flags *flags*, size_t *size*, void *host_ptr, cl_int *errcode_ret)

flags: See **clCreateBuffer**

cl_mem **clCreateSubBuffer** (
cl_mem *buffer*, cl_mem_flags *flags*, cl_buffer_create_type *buffer_create_type*, const void *buffer_create_info, cl_int *errcode_ret)

flags: See **clCreateBuffer**

buffer_create_type: CL_BUFFER_CREATE_TYPE_REGION

Read, write, copy, & fill buffer objects

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_read*, size_t *offset*, size_t *size*, void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_read*, const size_t *buffer_origin, const size_t *host_origin, const size_t *region, size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*, size_t *host_slice_pitch*, void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_write*, size_t *offset*, size_t *size*, const void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_write*, const size_t *buffer_origin, const size_t *host_origin, const size_t *region, size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*, size_t *host_slice_pitch*, const void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueFillBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, const void *pattern, size_t *pattern_size*, size_t *offset*, size_t *size*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBuffer** (
cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_buffer*, size_t *src_offset*, size_t *dst_offset*, size_t *size*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBufferRect** (
cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_buffer*, const size_t *src_origin, const size_t *dst_origin, const size_t *region, size_t *src_row_pitch*, size_t *src_slice_pitch*, size_t *dst_row_pitch*, size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

Map buffer objects

void * **clEnqueueMapBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_map*, cl_map_flags *map_flags*, size_t *offset*, size_t *size*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)

map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

Image Formats [API 5.3.1]

Image Channel Order Values [API Table 16]

CL_R	CL_RG	CL_RGBA	CL_RGBx
CL_A	CL_RA	CL_ARGB	CL_sRGB
CL_DEPTH	CL_Rx	CL_BGRA	CL_sRGBA
CL_LUMINANCE	CL_RGB	CL_ABGR	CL_sBGRA
CL_INTENSITY	CL_RGx	CL_ABGR	CL_sRGBx

Image Channel Data Types [API Table 17]

CL_SNORM_INT8	CL_SIGNED_INT8
CL_SNORM_INT16	CL_SIGNED_INT16
CL_UNORM_INT8	CL_SIGNED_INT32
CL_UNORM_INT16	CL_UNSIGNED_INT8
CL_UNORM_SHORT_565	CL_UNSIGNED_INT16
CL_UNORM_SHORT_555	CL_UNSIGNED_INT32
CL_UNORM_INT_101010	CL_HALF_FLOAT
CL_UNORM_INT_101010_2	CL_FLOAT

Image Objects [API 5.3]

Create image objects

cl_mem **clCreateImage** (
cl_context *context*, cl_mem_flags *flags*, const cl_image_format *image_format, const cl_image_desc *image_desc, void *host_ptr, cl_int *errcode_ret)

flags: CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_HOST_NO_ACCESS, CL_MEM_HOST_{READ, WRITE}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

cl_mem **clCreateImageWithProperties** (
cl_context *context*, const cl_mem_properties *properties, cl_mem_flags *flags*, const cl_image_format *image_format, const cl_image_desc *image_desc, void *host_ptr, cl_int *errcode_ret)

flags: See **clCreateImage**

Query list of supported image formats

cl_int **clGetSupportedImageFormats** (
cl_context *context*, cl_mem_flags *flags*, cl_mem_object_type *image_type*, cl_uint *num_entries*, cl_image_format *image_formats*, cl_uint *num_image_formats*)

flags: See **clCreateImage**

image_type: CL_MEM_OBJECT_IMAGE{1D, 2D, 3D}, CL_MEM_OBJECT_IMAGE1D_BUFFER, CL_MEM_OBJECT_IMAGE{1D, 2D}_ARRAY

Read, write, copy, & fill image objects

cl_int **clEnqueueReadImage** (
cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_read*, const size_t *origin, const size_t *region, size_t *row_pitch*, size_t *slice_pitch*, void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteImage** (
cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_write*, const size_t *origin, const size_t *region, size_t *input_row_pitch*, size_t *input_slice_pitch*, const void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueFillImage** (
cl_command_queue *command_queue*, cl_mem *image*, const void *fill_color, const size_t *origin, const size_t *region,cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyImage** (
cl_command_queue *command_queue*, cl_mem *src_image*, cl_mem *dst_image*, const size_t *src_origin, const size_t *dst_origin, const size_t *region, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

Copy between image & buffer objects

cl_int **clEnqueueCopyImageToBuffer** (
cl_command_queue *command_queue*, cl_mem *src_image*, cl_mem *dst_buffer*, const size_t *src_origin, const size_t *region, size_t *dst_offset*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBufferToImage** (
cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_image*, size_t *src_offset*, const size_t *dst_origin, const size_t *region, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

Map and unmap image objects

void * **clEnqueueMapImage** (
cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_map*, cl_map_flags *map_flags*, const size_t *origin, const size_t *region, size_t *image_row_pitch*, size_t *image_slice_pitch*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)

map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

Query image objects

cl_int **clGetImageInfo** (
cl_mem *image*, cl_image_info *param_name*, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name: CL_IMAGE_FORMAT, CL_IMAGE_{ARRAY, ELEMENT}_SIZE, CL_IMAGE_{ROW, SLICE}_PITCH, CL_IMAGE_{HEIGHT, WIDTH, DEPTH}, CL_IMAGE_NUM_{SAMPLES, MIP_LEVELS}

Pipes [API 5.4]

A pipe is a memory object that stores data organized as a FIFO. Pipe objects can only be accessed using built-in functions that read from and write to a pipe. Pipe objects are not accessible from the host.

Create pipe objects

cl_mem **clCreatePipe** (cl_context *context*,
cl_mem_flags *flags*, cl_uint *pipe_packet_size*,
cl_uint *pipe_max_packets*,
const cl_pipe_properties **properties*,
cl_int **errcode_ret*)

flags: 0 or CL_MEM_READ_WRITE, CL_MEM_HOST_NO_ACCESS

Pipe object queries

cl_int **clGetPipeInfo** (cl_mem *pipe*,
cl_pipe_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PIPE_PACKET_SIZE, CL_PIPE_MAX_PACKETS, CL_PIPE_PROPERTIES

Shared Virtual Memory [API 5.6]

Shared Virtual Memory (SVM) allows the host and kernels executing on devices to directly share complex, pointer-containing data structures such as trees and linked lists.

Allocate and free SVM

void* **clSVMAlloc** (
cl_context *context*, cl_svm_mem_flags *flags*,
size_t *size*, cl_uint *alignment*)

flags:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_SVM_FINE_GRAIN_BUFFER,
CL_MEM_SVM_ATOMICS

void **clSVMFree** (cl_context *context*, void **svm_pointer*)

SVM operations

cl_int **clEnqueueSVMFree** (
cl_command_queue *command_queue*,
cl_uint *num_svm_pointers*, void **sym_pointers*[],
void (CL_CALLBACK**pfn_free_func*)(
cl_command_queue *command_queue*,
cl_uint *num_svm_pointers*,
void **sym_pointers*[], void **user_data*),
void **user_data*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueSVMMemcpy** (
cl_command_queue *command_queue*,
cl_bool *blocking_copy*, void **dst_ptr*,
const void **src_ptr*, size_t *size*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueSVMMemFill** (
cl_command_queue *command_queue*,
void **svm_ptr*, const void **pattern*,
size_t *pattern_size*, size_t *size*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueSVMMap** (
cl_command_queue *command_queue*,
cl_bool *blocking_map*, cl_map_flags *map_flags*,
void **svm_ptr*, size_t *size*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueSVMUnmap** (
cl_command_queue *command_queue*,
void **svm_ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueSVMMigrateMem** (
cl_command_queue *command_queue*,
cl_uint *num_svm_pointers*, const void ***svm_pointers*,
const size_t **sizes*, cl_mem_migration_flags *flags*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Flush and Finish [API 5.15]

cl_int **clFlush** (cl_command_queue *command_queue*)
cl_int **clFinish** (cl_command_queue *command_queue*)

Memory Objects [API 5.5]

A memory object is a handle to a reference counted region of global memory. Includes buffer objects, image objects, and pipe objects.

Memory objects

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (
cl_mem *memobj*, void (CL_CALLBACK **pfn_notify*)
(cl_mem *memobj*, void **user_data*),
void **user_data*)

cl_int **clEnqueueUnmapMemObject** (
cl_command_queue *command_queue*,
cl_mem *memobj*, void **mapped_ptr*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Migrate memory objects

cl_int **clEnqueueMigrateMemObjects** (
cl_command_queue *command_queue*,
cl_uint *num_mem_objects*,
const cl_mem **mem_objects*,
cl_mem_migration_flags *flags*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query memory object

cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
cl_mem_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name:
CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_{CONTEXT, OFFSET, PROPERTIES},
CL_MEM_{MAP, REFERENCE}_COUNT,
CL_MEM_ASSOCIATED_MEMOBJECT,
CL_MEM_USES_SVM_POINTER

Sampler Objects [API 5.7]

cl_sampler
clCreateSamplerWithProperties (cl_context *context*,
const cl_sampler_properties **sampler_properties*,
cl_int **errcode_ret*)

sampler_properties:
CL_SAMPLER_NORMALIZED_COORDS,
CL_SAMPLER_{ADDRESSING, FILTER}_MODE

cl_int **clRetainSampler** (cl_sampler *sampler*)
cl_int **clReleaseSampler** (cl_sampler *sampler*)

cl_int **clGetSamplerInfo** (cl_sampler *sampler*,
cl_sampler_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)

param_name: CL_SAMPLER_REFERENCE_COUNT,
CL_SAMPLER_{CONTEXT, FILTER_MODE, PROPERTIES},
CL_SAMPLER_ADDRESSING_MODE,
CL_SAMPLER_NORMALIZED_COORDS

Sampler declaration fields [C 6.13.14]

The sampler can be passed as an argument to the kernel using **clSetKernelArg**, or declared in the outermost scope of kernel functions, or it can be a constant variable of type sampler_t declared in the program source.

const sampler_t <*sampler-name*> =
<*normalized-mode*> | <*address-mode*> | <*filter-mode*>

normalized-mode:
CLK_NORMALIZED_COORDS_{TRUE, FALSE}

address-mode:
CLK_ADDRESS_*X*, where *X* may be NONE, REPEAT, CLAMP, CLAMP_TO_EDGE, MIRRORED_REPEAT

filter-mode: CLK_FILTER_NEAREST, CLK_FILTER_LINEAR

Program Objects [API 5.8]

OpenCL programs consist of sets of kernels identified as functions declared with the __kernel qualifier in the program source.

Create program objects

cl_program **clCreateProgramWithSource** (
cl_context *context*, cl_uint *count*, const char ***strings*,
const size_t **lengths*, cl_int **errcode_ret*)

cl_program **clCreateProgramWithIL** (
cl_context *context*, const void **il*, size_t *length*,
cl_int **errcode_ret*)

cl_program **clCreateProgramWithBinary** (
cl_context *context*, cl_uint *num_devices*,
const cl_device_id **device_list*, const size_t **lengths*,
const unsigned char ***binaries*,
cl_int **binary_status*, cl_int **errcode_ret*)

cl_program **clCreateProgramWithBuiltInKernels** (
cl_context *context*, cl_uint *num_devices*,
const cl_device_id **device_list*,
const char **kernel_names*, cl_int **errcode_ret*)

Retain and release program objects

cl_int **clRetainProgram** (cl_program *program*)
cl_int **clReleaseProgram** (cl_program *program*)

cl_int **clSetProgramReleaseCallback**
(cl_program *program*, void (CL_CALLBACK**pfn_notify*)
(cl_program *prog*, void **user_data*),
void **user_data*)

Building program executables

cl_int **clBuildProgram** (cl_program *program*,
cl_uint *num_devices*, const cl_device_id **device_list*,
const char **options*, void (CL_CALLBACK**pfn_notify*)
(cl_program *program*, void **user_data*),
void **user_data*)

cl_int **clSetProgramSpecializationConstant**(
cl_program *program*, cl_uint *spec_id*,
size_t *spec_size*, const void* *spec_value*)

Separate compilation and linking

cl_int **clCompileProgram** (cl_program *program*,
cl_uint *num_devices*, const cl_device_id **device_list*,
const char **options*, cl_uint *num_input_headers*,
const cl_program **input_headers*,
const char ***header_include_names*,
void (CL_CALLBACK**pfn_notify*)
(cl_program *program*, void **user_data*),
void **user_data*)

cl_program **clLinkProgram** (cl_context *context*,
cl_uint *num_devices*, const cl_device_id **device_list*,
const char **options*, cl_uint *num_input_programs*,
const cl_program **input_programs*,
void (CL_CALLBACK**pfn_notify*)
(cl_program *program*, void **user_data*),
void **user_data*, cl_int **errcode_ret*)

Unload the OpenCL compiler

cl_int **clUnloadPlatformCompiler** (
cl_platform_id *platform*)

Query program objects

cl_int **clGetProgramInfo** (cl_program *program*,
cl_program_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)

param_name:
CL_PROGRAM_IL,
CL_PROGRAM_{REFERENCE_COUNT},
CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES},
CL_PROGRAM_{NUM_KERNELS, KERNEL_NAMES},
CL_PROGRAM_SCOPE_GLOBAL_CTORS_PRESENT,
CL_PROGRAM_SCOPE_GLOBAL_DTORS_PRESENT

cl_int **clGetProgramBuildInfo** (
cl_program *program*, cl_device_id *device*,
cl_program_build_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)

param_name:
CL_PROGRAM_BINARY_TYPE,
CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG},
CL_PROGRAM_BUILD_GLOBAL_VARIABLE_TOTAL_SIZE

(Continued on next page >)

Program Objects (continued)

Compiler options

Preprocessor:
(-D processed in order for **clBuildProgram** or **clCompileProgram**)
-D *name* -D *name=definition* -I dir

Math intrinsics:
-cl-single-precision-constant
-cl-denorms-are-zero
-cl-fp32-correctly-rounded-divide-sqrt

Optimization options:
-cl-opt-disable
-cl-no-signed-zeros
-cl-unsafe-math-optimizations
-cl-uniform-work-group-size
-cl-mad-enable
-cl-finite-math-only
-cl-fast-relaxed-math
-cl-no-subgroup-ifp

Warning request/suppress:
-w -Werror

Control OpenCL C language version:
-cl-std=CL1.1 OpenCL C 1.1 specification
-cl-std=CL1.2 OpenCL C 1.2 specification
-cl-std=CL2.0 OpenCL C 2.0 specification
-cl-std=CL3.0 OpenCL C 3.0 specification

Query kernel argument information:
-cl-kernel-arg-info

Debugging options:
-g Generate additional errors for built-in functions that allow you to enqueue commands on a device

Linker options

Library linking options:
-create-library -enable-link-options

Program linking options:
-cl-denorms-are-zero -cl-no-signed-zeroes
-cl-finite-math-only -cl-fast-relaxed-math
-cl-no-subgroup-ifp
-cl-unsafe-math-optimizations

Kernel Objects [API 5.9 - 5.10]

A kernel object encapsulates the specific __kernel function and the argument values to be used when executing it.

Create kernel objects

cl_kernel **clCreateKernel** (cl_program *program*, const char *kernel_name, cl_int *errcode_ret)

cl_int **clCreateKernelsInProgram** (cl_program *program*, cl_uint *num_kernels*, cl_kernel *kernels, cl_uint *num_kernels_ret)

cl_int **clRetainKernel** (cl_kernel *kernel*)

cl_int **clReleaseKernel** (cl_kernel *kernel*)

Kernel arguments and queries

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg_index*, size_t *arg_size*, const void *arg_value)

cl_int **clSetKernelArgSVMPointer** (cl_kernel *kernel*, cl_uint *arg_index*, const void *arg_value)

cl_int **clSetKernelExecInfo** (cl_kernel *kernel*, cl_kernel_exec_info *param_name*, size_t *param_value_size*, const void *param_value)

param_name: CL_KERNEL_EXEC_INFO_SVM_PTRS, CL_KERNEL_EXEC_INFO_SVM_FINE_GRAIN_SYSTEM

cl_kernel **clCloneKernel** (cl_kernel *source_kernel*, cl_int *errcode_ret)

cl_int **clGetKernelInfo** (cl_kernel *kernel*, cl_kernel_info *param_name*, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name:
CL_KERNEL_{FUNCTION_NAME, NUM_ARGS},
CL_KERNEL_REFERENCE_COUNT,
CL_KERNEL_{ATTRIBUTES, CONTEXT, PROGRAM}

cl_int **clGetKernelWorkGroupInfo** (cl_kernel *kernel*, cl_device_id *device*, cl_kernel_work_group_info *param_name*, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name: CL_KERNEL_GLOBAL_WORK_SIZE, CL_KERNEL_[COMPILE_]WORK_GROUP_SIZE, CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE, CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE

cl_int **clGetKernelArgInfo** (cl_kernel *kernel*, cl_uint *arg_indx*, cl_kernel_arg_info *param_name*, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name: CL_KERNEL_ARG_NAME, CL_KERNEL_ARG_{ACCESS, ADDRESS}_QUALIFIER, CL_KERNEL_ARG_TYPE_{NAME, QUALIFIER}

cl_int **clGetKernelSubGroupInfo** (cl_kernel *kernel*, cl_device_id *device*, cl_kernel_sub_group_info *param_name*, size_t *input_value_size*, const void *input_value, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name:
CL_KERNEL_LOCAL_SIZE_FOR_SUB_GROUP_COUNT,
CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_NDRANGE,
CL_KERNEL_SUB_GROUP_COUNT_FOR_NDRANGE
CL_KERNEL_MAX_NUM_SUB_GROUPS,
CL_KERNEL_COMPILE_NUM_SUB_GROUPS

Execute kernels

cl_int **clEnqueueNDRangeKernel** (cl_command_queue *command_queue*, cl_kernel *kernel*, cl_uint *work_dim*, const size_t *global_work_offset*, const size_t *global_work_size*, const size_t *local_work_size*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueNativeKernel** (cl_command_queue *command_queue*, void (CL_CALLBACK *user_func)(void *), void *args, size_t *cb_args*, cl_uint *num_mem_objects*, const cl_mem *mem_list, const void **args_mem_loc*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

NOTE: The ability to execute native kernels is optional within OpenCL and the semantics of native kernels are implementation-defined. The OpenCL API includes functions to query device capabilities and determine if this capability is supported.

Event Objects [API 5.11 - 5.14]

Event objects can be used to refer to a kernel execution command, and read, write, map, and copy commands on memory objects or user events.

Event objects

cl_event **clCreateUserEvent** (cl_context *context*, cl_int *errcode_ret)

cl_int **clSetUserEventStatus** (cl_event *event*, cl_int *execution_status*)

cl_int **clWaitForEvents** (cl_uint *num_events*, const cl_event *event_list)

cl_int **clGetEventInfo** (cl_event *event*, cl_event_info *param_name*, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name:
CL_EVENT_COMMAND_{QUEUE, TYPE},
CL_EVENT_{CONTEXT, REFERENCE_COUNT},
CL_EVENT_COMMAND_EXECUTION_STATUS

cl_int **clRetainEvent** (cl_event *event*)

cl_int **clReleaseEvent** (cl_event *event*)

cl_int **clSetEventCallback** (cl_event *event*, cl_int *command_exec_callback_type*, void (CL_CALLBACK *pfn_event_notify) (cl_event *event*, cl_int *event_command_exec_status*, void *user_data), void *user_data)

Markers, barriers, & waiting for events

cl_int **clEnqueueMarkerWithWaitList** (cl_command_queue *command_queue*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueBarrierWithWaitList** (cl_command_queue *command_queue*, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event)

Profiling operations

cl_int **clGetEventProfilingInfo** (cl_event *event*, cl_profiling_info *param_name*, size_t *param_value_size*, void *param_value, size_t *param_value_size_ret)

param_name:
CL_PROFILING_COMMAND_COMPLETE,
CL_PROFILING_COMMAND_QUEUED,
CL_PROFILING_COMMAND_{SUBMIT, START, END}

Memory Model: SVM [API 3.3.3]

OpenCL extends the global memory region into host memory through a shared virtual memory (SVM) mechanism. Three types of SVM in OpenCL:

- **Coarse-Grained buffer SVM**: Sharing at the granularity of regions of OpenCL buffer memory objects.
- **Fine-Grained buffer SVM**: Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects.
- **Fine-Grained system SVM**: Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory.

Summary of SVM options in OpenCL

SVM	Granularity of sharing	Memory allocation	Mechanisms to enforce consistency	Explicit updates between host and device?
Non-SVM buffers	OpenCL Memory objects (buffer)	clCreateBuffer	Host synchronization points on the same or between devices.	Yes, through Map and Unmap commands.
Coarse-Grained buffer SVM	OpenCL Memory objects (buffer)	clSVMAlloc	Host synchronization points between devices	Yes, through Map and Unmap commands.
Fine Grained buffer SVM	Bytes within OpenCL Memory objects (buffer)	clSVMAlloc	Synchronization points plus atomics (if supported)	No
Fine Grained system SVM	Bytes within Host memory (system)	Host memory allocation mechanisms (e.g. malloc)	Synchronization points plus atomics (if supported)	No

OpenCL C Language Reference

Section and table references are to the OpenCL 3.0 C Language specification.

Supported Data Types [C 6.1]

Double types require that CL_DEVICE_DOUBLE_FP_CONFIG is not zero.

Built-in Scalar Data Types

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
double	cl_double	64-bit IEEE 754
half	cl_half	16-bit float (storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	32- or 64-bit signed integer
uintptr_t	--	32- or 64-bit unsigned integer
void	void	void

Built-in Vector Data Types

n is 2, 3, 4, 8, or 16.

OpenCL Type	API Type	Description
[u]char*n*	cl_[u]char*n*	8-bit [un]signed
[u]short*n*	cl_[u]short*n*	16-bit [un]signed
[u]int*n*	cl_[u]int*n*	32-bit [un]signed
[u]long*n*	cl_[u]long*n*	64-bit [un]signed
float*n*	cl_float*n*	32-bit float
double*n*	cl_double*n*	64-bit float

Other Built-in Data Types

OpenCL Type	Description
event_t	event handle
queue_t	Requires support for OpenCL C 2.0 or the __opencl_c_device_enqueue feature macro.
ndrange_t	
clk_event_t	
reserve_id_t	Requires support for OpenCL C 2.0 or the__opencl_c_pipes feature macro.
cl_mem_fence_flags	

The following types shown below are only defined if the CL_DEVICE_IMAGE_SUPPORT device query is CL_TRUE.

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
image2d_array_t	2D image array
image1d_t	1D image handle
image1d_buffer_t	1D image buffer
image1d_array_t	1D image array
image2d_depth_t	2D depth image
image2d_array_depth_t	2D depth image array
sampler_t	sampler handle

Reserved Data Types

bool*n*
half*n*
quad, quad*n*
complex half, complex half*n* imaginary half, imaginary half*n*
complex float, complex float*n* imaginary float, imaginary float*n*
complex double, complex double*n* imaginary double, imaginary double*n*
complex quad, complex quad*n* imaginary quad, imaginary quad*n*
float*n*x*m*
double*n*x*m*
long double, long double*n*
long long, long long*n*
unsigned long long, ulong long, ulong long*n*

Vector Component Addressing [C 6.1.7]

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.s0	v.y, v.s1														
float3 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2													
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sA, v.sA	v.sB, v.sB	v.sC, v.sC	v.sD, v.sD	v.sE, v.sE	v.sF, v.sF

Vector Addressing Equivalences

Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3 *	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz

	v.lo	v.hi	v.odd	v.even
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

*When using .lo or .hi with a 3-component vector, the .w component is undefined.

Operators and Qualifiers

Operators [C 6.3]

These operators behave similarly as in C99 except operands may include vector types when possible:

+	-	*	%	/	--
++	==	!=	&	~	^
>	<	>=	<=	|	!
&&	||	?:	>>	<<	=
,	*op=*	*sizeof*			

Address Space Qualifiers [C 6.5]

__global, global __local, local
__constant, constant __private, private

Function Qualifiers [C 6.7]

__kernel, kernel
__attribute__((vec_type_hint(*type*)))
//*type* defaults to int
__attribute__((work_group_size_hint(*X*, *Y*, *Z*)))
__attribute__((reqd_work_group_size(*X*, *Y*, *Z*)))

Preprocessor Directives & Macros [C 6.10]

#pragma OPENCL FP_CONTRACT *on-off-switch*
on-off-switch: ON, OFF, DEFAULT

#pragma OPENCL EXTENSION *extensionname* : *behavior*

#pragma OPENCL EXTENSION all : *behavior*

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number, e.g: 300
CL_VERSION_1_0	Substitutes integer 100 for 1.0
CL_VERSION_1_1	Substitutes integer 110 for 1.1
CL_VERSION_1_2	Substitutes integer 120 for 1.2
CL_VERSION_2_0	Substitutes integer 200 for 2.0
CL_VERSION_3_0	Substitutes integer 300 for 3.0
__OPENCL_C_VERSION__	Sub. integer for OpenCL C version
__ENDIAN_LITTLE__	1 if device is little endian
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if –cl-fast-relaxed-math optimization option is specified

__kernel_exec (X, *typen*) Same as:
__kernel __attribute__((work_group_size_hint(*X*, 1, 1)))
__attribute__((vec_type_hint(*typen*)))

Conversions, Type Casting Examples [C 6.2]

T a = (*T*)*b*; *// Scalar to scalar, or scalar to vector*
T a = convert_*T*(*b*);
T a = convert_*T*_*R*(*b*);
T a = as_*T*(*b*);
T a = convert_*T*_sat_*R*(*b*);

R: one of the rounding modes
_rte to nearest even
_rtz toward zero
_rtp toward + infinity
_rtn toward - infinity

Attribute Qualifiers [C 6.11]

Use to specify special attributes of enum, struct, and union types.

__attribute__((aligned(*n*))) __attribute__((endian(host)))
__attribute__((aligned)) __attribute__((endian(device)))
__attribute__((packed)) __attribute__((endian))

Use to specify special attributes of variables or structure fields.

__attribute__((aligned(*alignment*)))

Use to specify basic blocks and control-flow-statements.

__attribute__((*attr1*)) {...}

Use to specify that a loop (for, while, and do loops) can be unrolled. (Must appear immediately before the loop to be affected.)

__attribute__((opencl_unroll_hint(*n*)))
__attribute__((opencl_unroll_hint))

Access Qualifiers [C 6.6]

Apply to 2D and 3D image types to declare if the image memory object is being read or written by a kernel.

__read_only, read_only __write_only, write_only

__read_write, read_write *(Requires OpenCL C 2.0 or the __opencl_c_read_write_images feature macro.)*

Blocks [C 6.12]

A result value type with a list of parameter types.
Requires support for the __opencl_c_device_enqueue feature macro or OpenCL C 2.0. For example:

1. The ^ declares variable "myBlock" is a Block.
2. The return type for the Block "myBlock"is int.
3. myBlock takes a single argument of type int.
4. The argument is named "num."
5. Multiplier captured from block's environment.

```
②    ①        ③
int (^myBlock)(int) =
 ^(int num) {return num * multiplier; };
                ④          ⑤
```

Work-Item Built-in Functions [C 6.13.1]

Query the number of dimensions, global, and local work size specified to **clEnqueueNDRangeKernel**, and global and local identifier of each work-item when this kernel is executed on a device. Funtions shown in **blue** require the feature macro __opencl_c_subgroups.

uint **get_work_dim** ()	Number of dimensions in use
size_t **get_global_size** (uint *dimindx*)	Number of global work-items
size_t **get_global_id** (uint *dimindx*)	Global work-item ID value
size_t **get_local_size** (uint *dimindx*)	Number of work-items in the work-group
size_t **get_enqueued_local_size** (uint *dimindx*)	Number of work-items in a uniform work-group
size_t **get_local_id** (uint *dimindx*)	Local work-item ID
size_t **get_num_groups** (uint *dimindx*)	Number of work-groups
size_t **get_group_id** (uint *dimindx*)	Work-group ID

size_t **get_global_offset** (uint *dimindx*)	Global offset
size_t **get_global_linear_id** ()	Work-items 1-dimensional global ID
size_t **get_local_linear_id** ()	Work-items 1-dimensional local ID
uint **get_sub_group_size** ()	Number of work-items in the subgroup
uint **get_max_sub_group_size** ()	Maximum size of a subgroup
uint **get_num_sub_groups** ()	Number of subgroups in the work-group
uint **get_enqueued_num_sub_groups** ()	Number of subgroups in a uniform work-group
uint **get_sub_group_id** ()	Sub-group ID
uint **get_sub_group_local_id** ()	Unique work-item ID

Math Built-in Functions [C 6.13.2]

The type used in a function must be the same for all arguments and the return type unless otherwise specified.

Ts is type float, optionally double (if double precision supported). *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*. All angles are in radians. *qual* may be __global, __local, or __private, or may be the generic address space with the __opencl_c_generic_address_space feature macro.

HN indicates that half and native variants are available using only the float or float*n* types by prepending "half_" or "native_" to the function name. Prototypes shown in brown text are available in half_ and native_ forms only using the float or float*n* types.

T **acos** (*T*)		Arc cosine
T **acosh** (*T*)		Inverse hyperbolic cosine
T **acospi** (*T x*)		acos (*x*) / π
T **asin** (*T*)		Arc sine
T **asinh** (*T*)		Inverse hyperbolic sine
T **asinpi** (*T x*)		asin (*x*) / π
T **atan** (*T y_over_x*)		Arc tangent
T **atan2** (*T y*, *T x*)		Arc tangent of *y* / *x*
T **atanh** (*T*)		Hyperbolic arc tangent
T **atanpi** (*T x*)		atan (*x*) / π
T **atan2pi** (*T x*, *T y*)		atan2 (*y*, *x*) / π
T **cbrt** (*T*)		Cube root
T **ceil** (*T*)		Round to integer toward + infinity
T **copysign** (*T x*, *T y*)		*x* with sign changed to sign of *y*
T **cos** (*T*)	HN	Cosine
T **cosh** (*T*)		Hyperbolic cosine
T **cospi** (*T x*)		cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)		*x* / *y* (*T* may only be float or float*n*)
T **erfc** (*T*)		Complementary error function
T **erf** (*T*)		Calculates error function of *T*
T **exp** (*T x*)	HN	Exponential base e
T **exp2** (*T*)	HN	Exponential base 2
T **exp10** (*T*)	HN	Exponential base 10
T **expm1** (*T x*)		e^x -1.0
T **fabs** (*T*)		Absolute value
T **fdim** (*T x*, *T y*)		Positive difference between *x* and *y*
T **floor** (*T*)		Round to integer toward infinity
T **fma** (*T a*, *T b*, *T c*)		Multiply and add, then round
T **fmax** (*T x*, *T y*) *Tn* **fmax** (*Tn x*, *Ts y*)		Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) *Tn* **fmin** (*Tn x*, *Ts y*)		Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)		Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *qual T *iptr*)		Fractional value in *x*
Ts **frexp** (*T x*, *qual* int **exp*) *Tn* **frexp** (*T x*, *qual* int*n* **exp*)		Extract mantissa and exponent

T **hypot** (*T x*, *T y*)		Square root of $x^2 + y^2$
int[*n*] **ilogb** (*T x*)		Return exponent as an integer value
Ts **ldexp** (*T x*, int *n*) *Tn* **ldexp** (*T x*, int*n n*)		$x * 2^n$
T **lgamma** (*T x*) *Ts* **lgamma_r** (*Ts x*, *qual* int **signp*) *Tn* **lgamma_r** (*Tn x*, *qual* int*n* **signp*)		Log gamma function
T **log** (*T*)	HN	Natural logarithm
T **log2** (*T*)	HN	Base 2 logarithm
T **log10** (*T*)	HN	Base 10 logarithm
T **log1p** (*T x*)		ln (1.0 + *x*)
T **logb** (*T x*)		Exponent of *x*
T **mad** (*T a*, *T b*, *T c*)		Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)		Maximum magnitude of *x* and *y*
T **minmag** (*T x*, *T y*)		Minimum magnitude of *x* and *y*
T **modf** (*T x*, *qual T *iptr*)		Decompose floating-point number
float[*n*] **nan** (uint[*n*] *nancode*)		Quiet NaN (Return is scalar when *nancode* is scalar)
double[*n*] **nan** (ulong[*n*] *nancode*)		Quiet NaN (Return is scalar when *nancode* is scalar)
T **nextafter** (*T x*, *T y*)		Next representable floating-point value after *x* in the direction of *y*
T **pow** (*T x*, *T y*)		Compute *x* to the power of *y*
Ts **pown** (*T x*, int *y*) *Tn* **pown** (*T x*, int*n y*)		Compute x^y, where *y* is an integer
T **powr** (*T x*, *T y*)	HN	Compute x^y, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)		1 / *x* (*T* may only be float or float*n*)
T **remainder** (*T x*, *T y*)		Floating point remainder
Ts **remquo** (*Ts x*, *Ts y*, *qual* int **quo*) *Tn* **remquo** (*Tn x*, *Tn y*, *qual* int*n* **quo*)		Remainder and quotient
T **rint** (*T*)		Round to nearest even integer
Ts **rootn** (*T x*, int *y*) *Tn* **rootn** (*T x*, int*n y*)		Compute *x* to the power of 1/*y*

T **round** (*T x*)		Integral value nearest to *x* rounding
T **rsqrt** (*T*)	HN	Inverse square root
T **sin** (*T*)	HN	Sine
T **sincos** (*T x*, *qual T *cosval*)		Sine and cosine of *x*
T **sinh** (*T*)		Hyperbolic sine
T **sinpi** (*T x*)		sin (π *x*)
T **sqrt** (*T*)	HN	Square root
T **tan** (*T*)	HN	Tangent
T **tanh** (*T*)		Hyperbolic tangent
T **tanpi** (*T x*)		tan (π *x*)
T **tgamma** (*T*)		Gamma function
T **trunc** (*T*)		Round to integer toward zero

Math Constants [C 6.13.2]

The values of the following symbolic constants are single-precision float.

MAXFLOAT	Value of maximum non-infinite single-precision floating-point number
HUGE_VALF	Positive float expression, evaluates to +infinity
HUGE_VAL	Positive double expression, evals. to +infinity (Requires double precision support.)
INFINITY	Constant float expression, positive or unsigned infinity
NAN	Constant float expression, quiet NaN

When double precision is supported, macros ending in _F are available in type double by removing _F from the macro name.

M_E_F	Value of e
M_LOG2E_F	Value of $\log_2 e$
M_LOG10E_F	Value of $\log_{10} e$
M_LN2_F	Value of $\log_e 2$
M_LN10_F	Value of $\log_e 10$
M_PI_F	Value of π
M_PI_2_F	Value of π / 2
M_PI_4_F	Value of π / 4
M_1_PI_F	Value of 1 / π
M_2_PI_F	Value of 2 / π
M_2_SQRTPI_F	Value of 2 / √π
M_SQRT2_F	Value of √2
M_SQRT1_2_F	Value of 1 / √2

Image Read and Write Functions [C 6.13.14]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage**. *sampler* specifies the addressing and filtering mode to use. *aQual* refers to one of the access qualifiers. For samplerless read functions this may be read_only or read_write.

Read and write functions for 2D images

Read an element from a 2D image, or write a color value to a location in a 2D image.

float4 **read_imagef** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

int4 **read_imagei** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

uint4 **read_imageui** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

float4 **read_imagef** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

int4 **read_imagei** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

uint4 **read_imageui** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

float **read_imagef** (read_only image2d_depth_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

float **read_imagef** (read_only image2d_array_depth_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

float4 **read_imagef** (*aQual* image2d_t *image*, int2 *coord*)

int4 **read_imagei** (*aQual* image2d_t *image*, int2 *coord*)

uint4 **read_imageui** (*aQual* image2d_t *image*, int2 *coord*)

float4 **read_imagef** (*aQual* image2d_array_t *image*, int4 *coord*)

int4 **read_imagei** (*aQual* image2d_array_t *image*, int4 *coord*)

uint4 **read_imageui** (*aQual* image2d_array_t *image*, int4 *coord*)

float **read_imagef** (*aQual* image2d_depth_t *image*, int2 *coord*)

float **read_imagef** (*aQual* image2d_array_depth_t *image*, int4 *coord*)

The **write_image{f, i, ui}** functions require support for OpenCL C 2.0 or the __opencl_c_3d_image_writes feature macro.

void **write_imagef** (*aQual* image2d_t *image*, int2 *coord*, float4 *color*)

void **write_imagei** (*aQual* image2d_t *image*, int2 *coord*, int4 *color*)

void **write_imageui** (*aQual* image2d_t *image*, int2 *coord*, uint4 *color*)

void **write_imagef** (*aQual* image2d_array_t *image*, int4 *coord*, float4 *color*)

void **write_imagei** (*aQual* image2d_array_t *image*, int4 *coord*, int4 *color*)

void **write_imageui** (*aQual* image2d_array_t *image*, int4 *coord*, uint4 *color*)

void **write_imagef** (*aQual* image2d_depth_t *image*, int2 *coord*, float *depth*)

void **write_imagef** (*aQual* image2d_array_depth_t *image*, int4 *coord*, float *depth*)

Read and write functions for 1D images

Read an element from a 1D image, or write a color value to a location in a 1D image.

float4 **read_imagef** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

int4 **read_imagei** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

uint4 **read_imageui** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

float4 **read_imagef** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float4} *coord*)

int4 **read_imagei** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

uint4 **read_imageui** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

float4 **read_imagef** (*aQual* image1d_t *image*, int *coord*)

float4 **read_imagef** (*aQual* image1d_buffer_t *image*, int *coord*)

int4 **read_imagei** (*aQual* image1d_t *image*, int *coord*)

uint4 **read_imageui** (*aQual* image1d_t *image*, int *coord*)

int4 **read_imagei** (*aQual* image1d_buffer_t *image*, int *coord*)

uint4 **read_imageui** (*aQual* image1d_buffer_t *image*, int *coord*)

float4 **read_imagef** (*aQual* image1d_array_t *image*, int2 *coord*)

int4 **read_imagei** (*aQual* image1d_array_t *image*, int2 *coord*)

uint4 **read_imageui** (*aQual* image1d_array_t *image*, int2 *coord*)

void **write_imagef** (*aQual* image1d_t *image*, int *coord*, float4 *color*)

void **write_imagei** (*aQual* image1d_t *image*, int *coord*, int4 *color*)

void **write_imageui** (*aQual* image1d_t *image*, int *coord*, uint4 *color*)

void **write_imagef** (*aQual* image1d_buffer_t *image*, int *coord*, float4 *color*)

void **write_imagei** (*aQual* image1d_buffer_t *image*, int *coord*, int4 *color*)

void **write_imageui** (*aQual* image1d_buffer_t *image*, int *coord*, uint4 *color*)

void **write_imagef** (*aQual* image1d_array_t *image*, int2 *coord*, float4 *color*)

void **write_imagei** (*aQual* image1d_array_t *image*, int2 *coord*, int4 *color*)

void **write_imageui** (*aQual* image1d_array_t *image*, int2 *coord*, uint4 *color*)

Read and write functions for 3D images

Read an element from a 3D image, or write a color value to a location in a 3D image.

float4 **read_imagef** (read_only image3d_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

int4 **read_imagei** (read_only image3d_t *image*, sampler_t *sampler*, int4 *coord)*

int4 **read_imagei** (read_only image3d_t *image*, sampler_t *sampler*, float4 *coord)*

uint4 **read_imageui** (read_only image3d_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

float4 **read_imagef** (*aQual* image3d_t *image*, int4 *coord*)

int4 **read_imagei** (*aQual* image3d_t *image*, int4 *coord*)

uint4 **read_imageui** (*aQual* image3d_t *image*, int4 *coord*)

Image Query Functions [C 6.13.14]

Query image width, height, and depth in pixels

int **get_image_width** (*aQual* image{1,2,3}d_t *image*)

int **get_image_width** (*aQual* image1d_buffer_t *image*)

int **get_image_width** (*aQual* image{1,2}d_array_t *image*)

int **get_image_width** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_height** (*aQual* image{2,3}d_t *image*)

int **get_image_height** (*aQual* image2d_array_t *image*)

int **get_image_height** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_depth** (image3d_t *image*)

Query image array size

size_t **get_image_array_size** (*aQual* image1d_array_t *image*)

size_t **get_image_array_size** (*aQual* image2d_array_t *image*)

size_t **get_image_array_size** (*aQual* image2d_array_depth_t *image*)

Query image dimensions

int2 **get_image_dim** (*aQual* image2d_t *image*)

int2 **get_image_dim** (*aQual* image2d_array_t *image*)

int4 **get_image_dim** (*aQual* image3d_t *image*)

int2 **get_image_dim** (*aQual* image2d_[array_]depth_t *image*)

Query image channel data type and order

int **get_image_channel_data_type** (*aQual* image{1,2,3}d_t *image*)

int **get_image_channel_data_type** (*aQual* image1d_buffer_t *image*)

int **get_image_channel_data_type** (*aQual* image{1,2}d_array_t *image*)

int **get_image_channel_data_type** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_channel_order** (*aQual* image{1,2,3}d_t *image*)

int **get_image_channel_order** (*aQual* image1d_buffer_t *image*)

int **get_image_channel_order** (*aQual* image{1,2}d_array_t *image*)

int **get_image_channel_order** (*aQual* image2d_[array_]depth_t *image*)

Common Built-in Functions [C 6.13.4]

These functions operate component-wise and use round to nearest even rounding mode. *Ts* is type float, optionally double (if double precision is supported). *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*.

T **clamp** (*T x*, *T min*, *T max*) *Tn* **clamp** (*Tn x*, *Ts min*, *Ts max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*) *Tn* **max** (*Tn x*, *Ts y*)	Max of *x* and *y*
T **min** (*T x*, *T y*) *Tn* **min** (*Tn x*, *Ts y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*) *Tn* **mix** (*Tn x*, *Tn y*, *Ts a*)	Linear blend of *x* and *y*
T **radians** (*T degrees*)	*degrees* to radians
T **step** (*T edge*, *T x*) *Tn* **step** (*Ts edge*, *Tn x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep** (*T edge0*, *T edge1*, *T x*) *T* **smoothstep** (*Ts edge0*, *Ts edge1*, *T x*)	Step and interpolate
T **sign** (*T x*)	Sign of *x*

Integer Built-in Functions [C 6.13.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*, where *n* is 2, 3, 4, 8, or 16. *Tu* is the unsigned version of *T*. *Tsc* is the scalar version of *T*.

Tu **abs** (*T x*)	| *x* |
Tu **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *Tsc min*, *Tsc max*)	min(max(*x*, *minval*), *maxval*)
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **ctz** (*T x*)	Number of trailing 0-bits in *x*
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *Tsc y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*) *T* **min** (*T x*, *Tsc y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	High half of the product of *x* and *y*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result
T **popcount** (*T x*)	Number of non-zero bits in *x*

*For **upsample**, return type is scalar when the parameters are scalar.*

short[*n*] **upsample** (char[*n*] *hi*, uchar[*n*] *lo*)	result[*i*]= ((short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort[*n*] **upsample** (uchar[*n*] *hi*, uchar[*n*] *lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int[*n*] **upsample** (short[*n*] *hi*, ushort[*n*] *lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint[*n*] **upsample** (ushort[*n*] *hi*, ushort[*n*] *lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long[*n*] **upsample** (int[*n*] *hi*, uint[*n*] *lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong[*n*] **upsample** (uint[*n*] *hi*, uint[*n*] *lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

The following fast integer functions optimize the performance of kernels. In these functions, *T* is type int, uint, int*n*, or uint*n*, where *n is 2, 3, 4, 8,* or *16*.

T **mad24** (*T x*, *T y*, *T z*)	Multiply 24-bit integer values *x*, *y*, add 32-bit int. result to 32-bit integer *z*
T **mul24** (*T x*, *T y*)	Multiply 24-bit integer values *x* and *y*

Geometric Built-in Functions [C 6.13.5]

Ts is scalar type float, optionally double (if double precision is supported). ***T*** is ***Ts*** and the 2-, 3-, or 4-component vector forms of ***Ts***.

float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*)	Cross product
Ts **distance** (*T p0*, *T p1*)	Vector distance
Ts **dot** (*T p0*, *T p1*)	Dot product
Ts **length** (*T p*)	Vector length
T **normalize** (*T p*)	Normal vector length 1
float **fast_distance** (float *p0*, float *p1*) float **fast_distance** (float*n* *p0*, float*n* *p1*)	Vector distance
float **fast_length** (float *p*) float **fast_length** (float*n* *p*)	Vector length
float **fast_normalize** (float *p*) float*n* **fast_normalize** (float*n* *p*)	Normal vector length 1

Relational Built-in Functions [C 6.13.6]

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result. *T* is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, or optionally double or double*n* (if double precision is supported). *Ti* is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. *Tu* is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. *n* is 2, 3, 4, 8, or 16.

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*)	Compare of *x* >= *y*
long*n* **isgreaterequal** (double*n* *x*, double*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*)	Compare of *x* < *y*
int **islessequal** (float *x*, float *y*) int*n* **islessequal** (float*n* *x*, float*n* *y*) int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*)	Compare of *x* <= *y*
int **islessgreater** (float *x*, float *y*) int*n* **islessgreater** (float*n* *x*, float*n* *y*) int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int **isfinite** (float) int*n* **isfinite** (float*n*) int **isfinite** (double) long*n* **isfinite** (double*n*)	Test for finite value
int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*)	Test for + or – infinity
int **isnan** (float) int*n* **isnan** (float*n*)	Test for a NaN
int **isnan** (double) long*n* **isnan** (double*n*)	Test for a NaN
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double)	Test for a normal value
long*n* **isnormal** (double*n*)	Test for a normal value
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* *y*)	Test if arguments are ordered
int **isunordered** (float *x*, float *y*) int*n* **isunordered** (float*n* *x*, float*n* *y*) int **isunordered** (double *x*, double *y*) long*n* **isunordered** (double*n* *x*, double*n* *y*)	Test if arguments are unordered
int **signbit** (float) int*n* **signbit** (float*n*) int **signbit** (double) long*n* **signbit** (double*n*)	Test for sign bit
int **any** (*Ti x*)	1 if MSB in component of x is set; else 0
int **all** (*Ti x*)	1 if MSB in all components of x are set; else 0
T **bitselect** (*T a*, *T b*, *T c*)	Each bit of result is corresponding bit of a if corresponding bit of c is 0
T **select** (*T a*, *T b*, *Ti c*) *T* **select** (*T a*, *T b*, *Tu c*)	For each component of a vector type, result[i] = if MSB of c[i] is set ? b[i] : a[i] For scalar type, result = c ? b : a

Vector Data Load/Store [C 6.13.7]

T is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double (if double precision is supported). *Tn* refers to vector form of type *T*, where *n* is 2, 3, 4, 8, or 16.
The default rounding mode is round to nearest even.
Load functions support global, local, private, and constant address spaces. Store functions support global, local, and private address spaces. For all, the generic address space may be supported with the __opencl_c_generic_address_space feature macro.

Tn **vload*n*** (size_t *offset*, const [constant] *T* *p)	Read vector data from address (*p* + (*offset* * *n*))
void **vstore*n*** (*Tn data*, size_t *offset*, *T* *p)	Write vector data to address (*p* + (*offset* * *n*)
float **vload_half** (size_t *offset*, const [constant] half *p)	Read a half from address (*p* + *offset*)
float*n* **vload_half*n*** (size_t *offset*, const [constant] half *p)	Read a half*n* from address (*p* + (*offset* * *n*))
void **vstore_half** (float *data*, size_t *offset*, half *p) void **vstore_half_*R*** (float *data*, size_t *offset*, half *p) void **vstore_half** (double *data*, size_t *offset*, half *p) void **vstore_half_*R*** (double *data*, size_t *offset*, half *p)	Write a half to address (*p* + *offset*)
void **vstore_half*n*** (float*n* *data*, size_t *offset*, half *p)	Write a half vector to address (*p* + (*offset* * *n*))
void **vstore_half*n*_*R*** (float*n* *data*, size_t *offset*, half *p)	Write a half vector to address (*p* + (*offset* * *n*))
void **vstore_half*n*** (double*n* *data*, size_t *offset*, half *p)	Write a half vector to address (*p* + (*offset* * *n*))
void **vstore_half*n*_*R*** (double*n* *data*, size_t *offset*, half *p)	Write a half vector to address (*p* + (*offset* * *n*))
void **vstorea_half*n*** (float*n* *data*, size_t *offset*, half *p)	Write half vector data to aligned (*p* + (*offset* * *n*)). For half3, write to aligned (*p* + (*offset* * 4)).
void **vstorea_half*n*_*R*** (float*n* *data*, size_t *offset*, half *p)	Write half vector data to aligned (*p* + (*offset* * *n*)). For half3, write to aligned (*p* + (*offset* * 4)).
void **vstorea_half*n*** (double*n* *data*, size_t *offset*, half *p)	Write half vector data to aligned (*p* + (*offset* * *n*)). For half3, write to aligned (*p* + (*offset* * 4)).
void **vstorea_half*n*_*R*** (double*n* *data*, size_t *offset*, half *p)	Write half vector data to aligned (*p* + (*offset* * *n*)). For half3, write to aligned (*p* + (*offset* * 4)).
void **vstorea_half*n*** (float*n* *data*, size_t *offset*, half *p)	Write half vector data to aligned (*p* + (*offset* * *n*)). For half3, write to aligned (*p* + (*offset* * 4)).

Synchronization & Memory Fence Functions [C 6.13.8]

flags argument is the memory address space, set to a 0 or an OR'd combination of CLK_*X*_MEM_FENCE where *X* may be LOCAL, GLOBAL, or IMAGE. Memory fence functions provide ordering between memory operations of a work-item.

void **barrier** (cl_mem_fence_flags *flags*) void **work_group_barrier** (cl_mem_fence_flags *flags* [, memory_scope *scope*])	Work-items in a work-group must execute this before any can continue.
void **sub_group_barrier** (cl_mem_fence_flags *flags* [, memory_scope *scope*])	Work-items in a sub-group must execute this before any can continue. Requires the feature macro __opencl_c_subgroups.

Miscellaneous Vector Functions [C 6.13.12]

Tm and *Tn* are type char*n*, uchar*n*, short*n*, ushort*n*, int*n*, uint*n*, long*n*, ulong*n*, float*n*, optionally double*n* (if double precision is supported), where *n* is 2,4,8, or 16 except in **vec_step** it may also be 3. *TUn* is uchar*n*, ushort*n*, uint*n*, or ulong*n*.

int **vec_step** (*Tn a*) int **vec_step** (*typename*)	Takes built-in scalar or vector data type argument. Returns 1 for scalar, 4 for 3-component vector, else number of elements in the specified type.
Tn **shuffle** (*Tm x, TUn mask*) *Tn* **shuffle2** (*Tm x, Tm y, TUn mask*)	Construct permutation of elements from one or two input vectors, return a vector with same element type as input and length that is the same as the shuffle mask.

Atomic Functions [C 6.13.11]

OpenCL C implements a subset of the C11 atomics (see section 7.17 of the C11 specification) and synchronization operations.

In the following tables, **A** refers to an atomic_* type (not including atomic_flag). **C** refers to its corresponding non-atomic type. **M** refers to the type of the other argument for arithmetic operations. For atomic integer types, **M** is **C.** For atomic pointer types, **M** is ptrdiff_t.

The atomic_double type is available if double precision is supported. The default scope is memory_scope_work_group for local atomics and memory_scope_device for global atomics.

The default scope is memory_scope_work_group for local atomics and memory_scope_device for global atomics, therefore the non-explicit functions require OpenCL C 2.0 or both the feature macros __opencl_c_atomic_order_seq_cst and __opencl_c_atomic_scope_device.

void **atomic_init**(volatile A *obj, C *value*)	Initializes the atomic object pointed to by *obj* to the value *value*.
void **atomic_work_item_fence**(cl_mem_fence_flags *flags*, memory_order *order*, memory_scope *scope*)	Effects based on value of *order*. *flags* must be CLK_{GLOBAL, LOCAL, IMAGE}_MEM_FENCE or a combination of these.
void **atomic_store**(volatile A *object, C *desired*) void **atomic_store_explicit**(volatile A *object, C *desired*, memory_order *order* [, memory_scope *scope*])	Atomically replace the value pointed to by *object* with the value of *desired*. Memory is affected according to the value of *order*.
C **atomic_load**(volatile A *object) C **atomic_load_explicit**(volatile A *object, memory_order *order*[, memory_scope *scope*])	Atomically returns the value pointed to by *object*. Memory is affected according to the value of *order*.
C **atomic_exchange**(volatile A *object, C *desired*) C **atomic_exchange_explicit**(volatile A *object, C *desired*, memory_order *order* [, memory_scope *scope*])	Atomically replace the value pointed to by *object* with *desired*. Memory is affected according to the value of *order*.
bool **atomic_compare_exchange_strong**(volatile A *object, C *expected, C *desired*) bool **atomic_compare_exchange_strong_explicit**(volatile A *object, C *expected, C *desired*, memory_order *success*, memory_order *failure*[, memory_scope *scope*]) bool **atomic_compare_exchange_weak**(volatile A *object, C *expected, C *desired*) bool **atomic_compare_exchange_weak_explicit**(volatile A *object, C *expected, C *desired*, memory_order *success*, memory_order *failure*[, memory_scope *scope*])	Atomically compares the value pointed to by *object* for equality with that in *expected*, and if true, replaces the value pointed to by *object* with *desired*, and if false, updates the value in *expected* with the value pointed to by *object*. These operations are atomic read-modify-write operations.
C **atomic_fetch_<*key*>**(volatile A *object, M *operand*) C **atomic_fetch_<*key*>_explicit**(volatile A *object, M *operand*, memory_order *order* [, memory_scope *scope*])	Atomically replaces the value pointed to by *object* with the result of the computation applied to the value pointed to by *object* and the given *operand*.

Async Copies and Prefetch [C 6.13.10]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, optionally double or double*n* (if double precision is supported).

event_t **async_work_group_copy** (__local *T* *dst, const __global *T* *src, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global *T* *dst, const __local *T* *src, size_t *num_gentypes*, event_t *event*)	Copies *num_gentypes* *T* elements from *src* to *dst*
event_t **async_work_group_strided_copy** (__local *T* *dst, const __global *T* *src, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global *T* *dst, const __local *T* *src, size_t *num_gentypes*, size_t *dst_stride*, event_t *event*)	
void **wait_group_events** (int *num_events*, event_t *event_list*)	Wait for completion of **async_work_group_copy**
void **prefetch** (const __global *T* *p, size_t *num_gentypes*)	Prefetch *num_gentypes* * sizeof(*T*) bytes into global cache

Address Space Qualifier Functions [C 6.13.9]

T refers to any of the built-in data types supported by OpenCL C or a user-defined type. These functions require the __opencl_c_generic_address_space feature macro.

[const] global *T* * **to_global** ([const] *T* *ptr)	global address space
[const] local *T* * **to_local** ([const] *T* *ptr)	local address space
[const] private *T* * **to_private** ([const] *T* *ptr)	private address space
[const] cl_mem_fence_flags **get_fence** ([const] T *ptr)	Memory fence value: CLK_GLOBAL_MEM_FENCE, CLK_IMAGE_MEM_FENCE, CLK_LOCAL_MEM_FENCE

bool **atomic_flag_test_and_set**(volatile atomic_flag *object) bool **atomic_flag_test_and_set_explicit**(volatile atomic_flag *object, memory_order *order*[, memory_scope *scope*])	Atomically sets the value pointed to by *object* to true. Memory is affected according to the value of *order*. Returns atomically, the value of the object immediately before the effects.
void **atomic_flag_clear**(volatile atomic_flag *object) void **atomic_flag_clear_explicit**(volatile atomic_flag *object, memory_order *order*[, memory_scope *scope*])	Atomically sets the value pointed to by *object* to false. The order argument shall not be *memory_order_acquire* nor *memory_order_acq_rel*. Memory is affected according to the value of *order*.

Values for *key* for atomic_fetch and modify functions

key	op	computation
add	+	addition
sub	-	subtraction
or	|	bitwise inclusive or
xor	^	bitwise exclusive or

key	op	computation
and	&	bitwise and
min	min	compute min
max	max	compute max

Atomic Types and Enum Constants

Parameter type: memory_order

Values	Optional requirements
memory_order_relaxed	
memory_order_acquire	With any built-in atomic function except **atomic_work_item_fence**, requires OpenCL C 2.0 or __opencl_c_atomic_order_acq_rel
memory_order_release	
memory_order_ acq_rel	
memory_order_seq_cst	Requires OpenCL C 2.0 or __opencl_c_atomic_order_seq_cst

Parameter type: memory_scope

Values	Optional requirements
memory_scope_work_group	
memory_scope_work_item	Only used with **atomic_work_item_fence** with flags: CLK_IMAGE_MEM_FENCE
memory_scope_sub_group	Requires __opencl_c_subgroups
memory_scope_device	Requires OpenCL C 2.0 or __opencl_c_atomic_scope_device
memory_scope_all_svm_devices	Requires OpenCL C 2.0 or __opencl_c_atomic_scope_all_svm_devices

(Continued on next page >)

Atomic Functions (continued)

Atomic macros

#define ATOMIC_VAR_INIT(C value)

Expands to a token sequence to initialize an atomic object of a type that is initialization-compatible with value.

#define ATOMIC_FLAG_INIT

Global atomic objects declared with the atomic_flag type can be initialized to a clear state with the ATOMIC_FLAG_INIT macro, for example:

```
global atomic_flag guard = ATOMIC_FLAG_INIT;
```

Atomic integer and floating-point types

† indicates types supported by a limited subset of atomic operations.

‡ indicates size depends on whether implemented on 64-bit or 32-bit architecture.

§ indicates types supported only with these extensions enabled: *cl_khr_int64_base_atomics* and *cl_khr_int64_extended_atomics*

atomic_int	atomic_double †§	atomic_ptrdiff_t ‡§
atomic_uint	atomic_long §	atomic_intptr_t ‡§
atomic_flag	atomic_ulong §	atomic_uintptr_t ‡§
atomic_float †	atomic_size_t ‡§	

Legacy Atomic Functions

These functions functions provide atomic operations on 32-bit signed and unsigned integers and single precision floating-point to locations in __global or __local memory. ***T*** is type int or unsigned int. ***T*** may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. ***Q*** is volatile __global or volatile __local.

***T* atomic_add** (*Q T *p, T val*)	Read, add, and store
***T* atomic_sub** (*Q T *p, T val*)	Read, subtract, and store
***T* atomic_xchg** (*Q T *p, T val*)	Read, swap, and store
***T* atomic_inc** (*Q T *p*)	Read, increment, and store
***T* atomic_dec** (*Q T *p*)	Read, decrement, and store
***T* atomic_cmpxchg** (*Q T *p, T cmp, T val*)	Read, store (**p ==cmp*) ? *val* : **p*
***T* atomic_min** (*Q T *p, T val*)	Read, store min(**p, val*)
***T* atomic_max** (*Q T *p, T val*)	Read, store max(**p, val*)
***T* atomic_and** (*Q T *p, T val*)	Read, store (**p* & *val*)
***T* atomic_or** (*Q T *p, T val*)	Read, store (**p* | *val*)
***T* atomic_xor** (*Q T *p, T val*)	Read, store (**p* ^ *val*)

printf Function [C 6.13.13]

Writes output to an implementation-defined stream.

int **printf** (constant char * restrict *format*, ...)

printf output synchronization

When the event associated with a particular kernel invocation completes, the output of applicable **printf** calls is flushed to the implementation-defined output stream.

printf format string

The format string follows C99 conventions and supports an optional vector specifier:

%[*flags*][*width*][.*precision*][*vector*][*length*] *conversion*

Examples:

The following examples show the use of the vector specifier in the **printf** format string.

```
float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);
uchar4 uc = (uchar4)(0xFA, 0xFB, 0xFC, 0xFD);
printf("f4 = %2.2v4hlf\n", f);
printf("uc = %#v4hhx\n", uc);
```

The above two printf calls print the following:

```
f4 = 1.00,2.00,3.00,4.00
uc = 0xfa,0xfb,0xfc,0xfd
```

Work-group Functions [C 6.13.15]

T is type int, uint, long, ulong, or float, optionally double (if double precision is supported). The **sub_group_*** work-group functions require the feature macro __opencl_c_subgroups. All other work-group functions require OpenCL C 2.0 or __opencl_c_work_group_collective_functions.

Returns a non-zero value if *predicate* evaluates to non-zero for all or any work-items in the work-group.

int **work_group_all** (int *predicate*)
int **work_group_any** (int *predicate*)
int **sub_group_all** (int *predicate*)
int **sub_group_any** (int *predicate*)

Return result of reduction operation specified by <*op*> for all values of *x* specified by work-items in work-group. <*op*> may be min, max, or add.

T **work_group_reduce_<*op*>** (*T x*)
T **sub_group_reduce_<op>** (*T x*)

Broadcast the value of *a* to all work-items in the work-group. *local_id* must be the same value for all work-items in the work-group.

T **work_group_broadcast** (*T* a, size_t *local_id*)
T **work_group_broadcast** (*T* a, size_t *local_id_x*, size_t *local_id_y*)
T **work_group_broadcast** (*T* a, size_t *local_id_x*, size_t *local_id_y*, size_t *local_id_z*)
T **sub_group_broadcast** (*T* x, size_t *local_id*)

Do an exclusive or inclusive scan operation specified by <*op*> of all values specified by work-items in the work-group. The scan results are returned for each work-item. <*op*> may be min, max, or add.

T **work_group_scan_exclusive_<op>** (*T x*)
T **work_group_scan_inclusive_<op>** (*T x*)
T **sub_group_scan_exclusive_<op>** (*T x*)
T **sub_group_scan_inclusive_<op>** (*T x*)

Pipe Built-in Functions [C 6.13.16]

T represents the built-in OpenCL C scalar or vector integer or floating-point data types or any user defined type built from these scalar and vector data types. Double or vector double types require double precision to be supported. The macro CLK_NULL_RESERVE_ID refers to an invalid reservation ID.

The **sub_group_*** pipe functions require the feature macro __opencl_c_subgroups. All other functions require __opencl_c_pipes or OpenCL C 2.0.

int **read_pipe** (__read_only pipe *T p*, reserve_id_t *reserve_id*, uint *index*, *T *ptr*)	Read packet from *p* into *ptr*.
int **read_pipe** (__read_only pipe *T p*, reserve_id_t *reserve_id*, uint *index*, *T *ptr*)	Read packet from reserved area of the pipe *reserve_id* and *index* into *ptr*.
int **write_pipe** (__write_only pipe *T p*, const *T *ptr*)	Write packet specified by *ptr* to *p*.
int **write_pipe** (__write_only pipe *T p*, reserve_id_t *reserve_id*, uint *index*, const *T *ptr*)	Write packet specified by *ptr* to reserved area *reserve_id* and *index*.

bool **is_valid_reserve_id** (reserve_id_t *reserve_id*)	Return true if *reserve_id* is a valid reservation ID and false otherwise.
reserve_id_t **reserve_read_pipe** (__read_only pipe *T p*, uint *num_packets*) reserve_id_t **reserve_write_pipe** (__write_only pipe *T p*, uint *num_packets*)	Reserve *num_packets* entries for reading from or writing to *p*.
void **commit_read_pipe** (__read_only pipe *T p*, reserve_id_t *reserve_id*) void **commit_write_pipe** (__write_only pipe *T p*, reserve_id_t *reserve_id*)	Indicates that all reads and writes to *num_packets* associated with reservation *reserve_id* are completed.
uint **get_pipe_max_packets** (pipe *T p*)	Returns maximum number of packets specified when *p* was created.
uint **get_pipe_num_packets** (pipe *T p*)	Returns the number of available entries in *p*.

void **work_group_commit_read_pipe** (pipe *T p*, reserve_id_t *reserve_id*) void **work_group_commit_write_pipe** (pipe *T p*, reserve_id_t *reserve_id*) void **sub_group_commit_read_pipe** (pipe *T p*, reserve_id_t *reserve_id*) void **sub_group_commit_write_pipe** (pipe *T p*, reserve_id_t *reserve_id*)	Indicates that all reads and writes to *num_packets* associated with reservation *reserve_id* are completed.
reserve_id_t **work_group_reserve_read_pipe** (pipe *T p*, uint *num_packets*) reserve_id_t **work_group_reserve_write_pipe** (pipe *T p*, uint *num_packets*) reserve_id_t **sub_group_reserve_read_pipe** (pipe *T p*, uint *num_packets*) reserve_id_t **sub_group_reserve_write_pipe** (pipe *T p*, uint *num_packets*)	Reserve *num_packets* entries for reading from or writing to *p*. Returns a valid reservation ID if the reservation is successful.

Notes

Enqueuing and Kernel Query Built-in Functions [C 6.13.17]

A kernel may enqueue code represented by Block syntax, and control execution order with event dependencies including user events and markers. There are several advantages to using the Block syntax: it is more compact; it does not require a cl_kernel object; and enqueuing can be done as a single semantic step. The macro CLK_NULL_EVENT refers to an invalid device event. The macro CLK_NULL_QUEUE refers to an invalid device queue.

The ***_sub_group_*** functions require support for the feature macros __opencl_c_subgroups and __opencl_c_device_enqueue. All other functions require support for __opencl_c_device_enqueue or OpenCL C 2.0.

int **enqueue_kernel** (queue_t *queue*, kernel_enqueue_flags_t *flags*, const ndrange_t *ndrange*, void (^*block*)(void)) int **enqueue_kernel** (queue_t *queue*, kernel_enqueue_flags_t *flags*, const ndrange_t *ndrange*, uint *num_events_in_wait_list*, const clk_event_t **event_wait_list*, clk_event_t **event_ret*, void (^*block*)(void)) int **enqueue_kernel** (queue_t *queue*, kernel_enqueue_flags_t *flags*, const ndrange_t *ndrange*, void (^*block*)(local void *, ...), uint *size0*, ...) int **enqueue_kernel** (queue_t *queue*, kernel_ enqueue_flags_t *flags*, const ndrange_t *ndrange*, uint *num_events_in_wait_list*, const clk_event_t **event_wait_list*, clk_event_t **event_ret*, void (^*block*)(local void *, ...), uint *size0*, ...)	Allows a work-item to enqueue a block for execution to *queue*. Work-items can enqueue multiple blocks to a device queue(s). *flags* may be one of CLK_ENQUEUE_FLAGS_{NO_WAIT, WAIT_KERNEL, WAIT_WORK_GROUP}

uint **get_kernel_work_group_size** (void (^*block*)(void)) uint **get_kernel_work_group_size** (void (^*block*)(local void *, ...))	Query the maximum work-group size that can be used to execute a block.
uint **get_kernel_preferred_work_group_size_multiple** (void (^*block*)(void)) uint **get_kernel_preferred_work_group_size_multiple** (void (^*block*)(local void *, ...))	Returns the preferred multiple of work-group size for launch.
int **enqueue_marker** (queue_t *queue*, uint *num_events_in_wait_list*, const clk_event_t **event_wait_list*, clk_event_t **event_ret*)	Enqueue a marker command to *queue*.
uint **get_kernel_sub_group_count_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(void)) uint **get_kernel_sub_group_count_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(local void *, ...))	Returns number of subgroups in each work-group of the dispatch.
uint **get_kernel_max_sub_group_size_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(void)) uint **get_kernel_max_sub_group_size_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(local void *, ...))	Returns the maximum sub-group size for a block.

Event Built-in Functions [C 6.13.17]

T is type int, uint, long, ulong, or float, optionally double (if double precision is supported). These functions require support for the __opencl_c_device_enqueue feature macro or OpenCL C 2.0.

void **retain_event** (clk_event_t *event*)	Increments event reference count.
void **release_event** (clk_event_t *event*)	Decrements event reference count.
clk_event_t **create_user_event** ()	Create a user event.
bool **is_valid_event** (clk_event_t *event*)	True for valid event.
void **set_user_event_status** (clk_event_t *event*, int *status*)	Sets the execution status of a user event. *status*: CL_COMPLETE or a negative error value.
void **capture_event_profiling_info** (clk_event_t *event*, clk_profiling_info *name*, global void **value*)	Captures profiling information for command associated with *event* in value.

Helper Built-in Functions [C 6.13.17]

These functions require support for the __opencl_c_device_enqueue feature macro or OpenCL C 2.0.

queue_t **get_default_queue** (void)	Default queue or CLK_NULL_QUEUE
ndrange_t **ndrange_1D** (size_t *global_work_size*) ndrange_t **ndrange_1D** (size_t *global_work_size*, size_t *local_work_size*) ndrange_t **ndrange_1D** (size_t *global_work_offset*, size_t *global_work_size*, size_t *local_work_size*)	Builds a 1D ND-range descriptor.
ndrange_t **ndrange_*n*D** (const size_t *global_work_size*[*n*]) ndrange_t **ndrange_*n*D** (size_t *global_work_size*, const size_t *local_work_size*[*n*]) ndrange_t **ndrange_*n*D** (const size_t *global_work_offset*, const size_t *global_work_size*, const size_t *local_work_size*[*n*])	Builds a 2D or 3D ND-range descriptor. *n* may be 2 or 3.

Feature Macros [C Appendix A]

When an OpenCL C optional feature is supported in the language, support will be indicated using a feature macro.

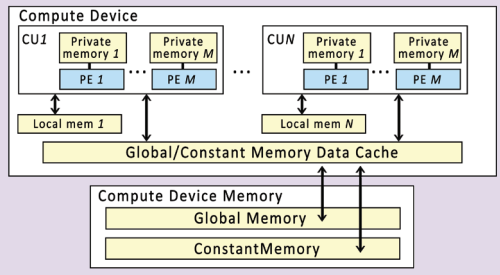
Feature Macro	The OpenCL C compiler supports...
__opencl_c_3d_image_writes	Built-in functions for writing to 3D image objects.
__opencl_c_atomic_order_acq_rel	Enumerations and built-in functions for atomic operations with acquire and release memory consistency orders.
__opencl_c_atomic_order_seq_cst	Enumerations and built-in functions for atomic operations and fences with sequentially consistent memory consistency order.
__opencl_c_atomic_scope_device	Enumerations and built-in functions for atomic operations and fences with device memory scope.
__opencl_c_atomic_scope_all_svm_devices	Enumerations and built-in functions for atomic operations and fences with all SVM devices memory scope.
__opencl_c_device_enqueue	Built-in functions to enqueue additional work from the device.
__opencl_c_generic_address_space	The unnamed generic address space.
__opencl_c_pipes	The pipe modifier and built-in functions to read and write from a pipe.
__opencl_c_program_scope_global_variables	Program scope variables in the global address space.
__opencl_c_read_write_images	Reading from and writing to the same image object in a kernel.
__opencl_c_subgroups	Built-in functions operating on sub-groupings of work-items.
__opencl_c_work_group_collective_functions	Built-in functions that perform collective operations across a work-group.

OpenCL Device Architecture Diagram

The table below shows memory regions with allocation and memory access capabilities.

	Global	Constant	Local	Private
Host	Dynamic allocation R/W access	Dynamic allocation R/W access	Dynamic allocation No access	No allocation No access
Kernel	No allocation R/W access	Static allocation R-only access	Static allocation R/W access	Static allocation R/W access

The conceptual OpenCL device architecture diagram shows processing elements (PE), compute units (CU), and devices. The host is not shown.



OpenCL™ | KHRONOS GROUP

The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See www.khronos.org to learn more about the Khronos Group.