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]

- `clGetPlatformIDs` ([cl_platform_id *platforms], [cl_uint *num_platforms])
- `clGetPlatformInfo` ([cl_platform_id *platform], [cl_platform_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])

- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Contexts [API 4.4]

- `clCreateContext` ([const char *extensions], [void *private_info], [size_t *size], [void *user_data], [CL_CONTEXT_PLATFORM], [void *cb])
- `clGetExtensionFunctionAddressForPlatform` ([const char *fn_name], [const char *extension], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Partitioning a device [API 4.3]

- `clGetDeviceSubDevices` ([cl_device_id *device], [CL_DEVICE_PARTITION_PROPERTY *properties], [cl_uint *num_devices], [void *out_devices], [cl_uint *num_devices_ret])
- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])
- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Get CL extension function pointers [Ext 1.3]

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

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]

- `clCreateCommandQueueWithProperties` ([cl_context *context], [cl_device_id *device], [const cl_command_queue_properties *properties], [cl_int *error_code])

Properties: NULL or a pointer to a zero-terminated list of properties and their values:

- `[cl_queue_properties *properties]`
- `[cl_command_queue_properties *properties]`

Get OpenCL extension information

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

OpenCL 3.0 (Provisional) Reference Guide

Content relating to optional features in OpenCL 3.0

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]

- `clGetPlatformIDs` ([cl_platform_id *platforms], [cl_uint *num_platforms])
- `clGetPlatformInfo` ([cl_platform_id *platform], [cl_platform_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])

- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Contexts [API 4.4]

- `clCreateContext` ([const char *extensions], [void *private_info], [size_t *size], [void *user_data], [CL_CONTEXT_PLATFORM], [void *cb])
- `clGetExtensionFunctionAddressForPlatform` ([const char *fn_name], [const char *extension], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Partitioning a device [API 4.3]

- `clGetDeviceSubDevices` ([cl_device_id *device], [CL_DEVICE_PARTITION_PROPERTY *properties], [cl_uint *num_devices], [void *out_devices], [cl_uint *num_devices_ret])
- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])
- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Get CL extension function pointers [Ext 1.3]

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

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]

- `clCreateCommandQueueWithProperties` ([cl_context *context], [cl_device_id *device], [const cl_command_queue_properties *properties], [cl_int *error_code])

Properties: NULL or a pointer to a zero-terminated list of properties and their values:

- `[cl_queue_properties *properties]`
- `[cl_command_queue_properties *properties]`

Get OpenCL extension information

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

OpenCL 3.0 (Provisional) Reference Guide

Content relating to optional features in OpenCL 3.0

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]

- `clGetPlatformIDs` ([cl_platform_id *platforms], [cl_uint *num_platforms])
- `clGetPlatformInfo` ([cl_platform_id *platform], [cl_platform_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])

- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Contexts [API 4.4]

- `clCreateContext` ([const char *extensions], [void *private_info], [size_t *size], [void *user_data], [CL_CONTEXT_PLATFORM], [void *cb])
- `clGetExtensionFunctionAddressForPlatform` ([const char *fn_name], [const char *extension], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Partitioning a device [API 4.3]

- `clGetDeviceSubDevices` ([cl_device_id *device], [CL_DEVICE_PARTITION_PROPERTY *properties], [cl_uint *num_devices], [void *out_devices], [cl_uint *num_devices_ret])
- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])
- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Get CL extension function pointers [Ext 1.3]

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

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]

- `clCreateCommandQueueWithProperties` ([cl_context *context], [cl_device_id *device], [const cl_command_queue_properties *properties], [cl_int *error_code])

Properties: NULL or a pointer to a zero-terminated list of properties and their values:

- `[cl_queue_properties *properties]`
- `[cl_command_queue_properties *properties]`

Get OpenCL extension information

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

OpenCL 3.0 (Provisional) Reference Guide

Content relating to optional features in OpenCL 3.0

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]

- `clGetPlatformIDs` ([cl_platform_id *platforms], [cl_uint *num_platforms])
- `clGetPlatformInfo` ([cl_platform_id *platform], [cl_platform_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])

- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Contexts [API 4.4]

- `clCreateContext` ([const char *extensions], [void *private_info], [size_t *size], [void *user_data], [CL_CONTEXT_PLATFORM], [void *cb])
- `clGetExtensionFunctionAddressForPlatform` ([const char *fn_name], [const char *extension], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Partitioning a device [API 4.3]

- `clGetDeviceSubDevices` ([cl_device_id *device], [CL_DEVICE_PARTITION_PROPERTY *properties], [cl_uint *num_devices], [void *out_devices], [cl_uint *num_devices_ret])
- `clGetDeviceIDs` ([cl_platform_id *platform], [cl_device_type *device_type], [cl_uint *num_devices])
- `clGetDeviceInfo` ([cl_device_id *device], [cl_device_info *param_name], [size_t *param_value_size], [void *param_value], [size_t *param_value_size_ret])

Get CL extension function pointers [Ext 1.3]

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])

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]

- `clCreateCommandQueueWithProperties` ([cl_context *context], [cl_device_id *device], [const cl_command_queue_properties *properties], [cl_int *error_code])

Properties: NULL or a pointer to a zero-terminated list of properties and their values:

- `[cl_queue_properties *properties]`
- `[cl_command_queue_properties *properties]`

Get OpenCL extension information

- `clGetExtensionFunctionAddressForPlatform` ([cl_platform_id *platform], [const char *fnname])
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

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

```c
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)
```

```c
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)
```

Read, write, copy, & fill buffer objects

```c
cl_int clEnqueueFillBuffer (cl_command_queue command_queue, cl_mem buffer, cl_buffer_copy_type buffer_copy_type,
    cl_mem_flags flags, size_t offset, size_t size, cl_uint num_events_in_wait_list,
    cl_event *event_wait_list, cl_event *event)
```

```c
cl_int clEnqueueWriteImageBuffer (cl_command_queue command_queue, cl_mem buffer, cl_buffer_write_type buffer_write_type,
    cl_mem_flags flags, size_t offset, size_t size, cl_uint num_events_in_wait_list,
    cl_event *event_wait_list, cl_event *event)
```

Image Formats [API 5.3.1]

**Image Channel Order Values [API Table 16]**

<table>
<thead>
<tr>
<th>CL_R</th>
<th>CL_G</th>
<th>CL_B</th>
<th>CL_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_RGBA</td>
<td>CL_RGBA</td>
<td>CL_RGBA</td>
<td>CL_RGBA</td>
</tr>
<tr>
<td>CL_RGB</td>
<td>CL_RGB</td>
<td>CL_RGB</td>
<td>CL_RGB</td>
</tr>
<tr>
<td>CL_LUMINANCE</td>
<td>CL_LUMINANCE</td>
<td>CL_LUMINANCE</td>
<td>CL_LUMINANCE</td>
</tr>
<tr>
<td>CL_INTENSITY</td>
<td>CL_INTENSITY</td>
<td>CL_INTENSITY</td>
<td>CL_INTENSITY</td>
</tr>
<tr>
<td>CL_R8</td>
<td>CL_G8</td>
<td>CL_B8</td>
<td>CL_A8</td>
</tr>
<tr>
<td>CL_RGB8</td>
<td>CL_RGB8</td>
<td>CL_RGB8</td>
<td>CL_RGB8</td>
</tr>
<tr>
<td>CL_RGBA8</td>
<td>CL_RGBA8</td>
<td>CL_RGBA8</td>
<td>CL_RGBA8</td>
</tr>
</tbody>
</table>

**Image Objects [API 5.3]**

Create image objects

```c
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)
```

```c
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)
```

Query list of supported image formats

```c
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)
```

Map and unmap image objects

```c
void *clEnqueueMapImage (cl_command_queue command_queue, cl_mem src_image, cl_mem dst_image,
    cl_image_layout layout, cl_int num_segments, size_t origin, size_t region,
    cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
```

Copy between image & buffer objects

```c
cl_int clEnqueueCopyImageToBuffer (cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer,
    cl_copy_mode mode, cl_buffer_copy_type buffer_copy_type, cl_mem_flags flags,
    const void *ptr_row_start, size_t row_pitch, size_t slice_pitch, const void *ptr_dst_row_start,
    const size_t *region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
```

Query image objects

```c
cl_int clGetImageInfo (cl_context context, cl_mem_image image, cl_image_info info_type,
    size_t info_size, const void *info_value, cl_int *info_value_size)
```
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
c_l_int clRetainMemObject (cl_mem memobj)
c_l_int clReleaseMemObject (cl_mem memobj)
c_l_int clSetMemObjectDestructorCallback (cl_mem memobj, void (CL_CALLBACK * pfn_notify)(cl_mem, void *user_data), void *user_data)

clSVM operations
c_l_void* clSVMAlloc (const size_t alignment, cl_uint num_svm_pointers, const size_t * sizes, cl_svm_memflags flags, cl_uint num_events_in_wait_list, const void * src_ptr, size_t src_size, cl_int * errcode_ret)
c_l_int clSVMFree (cl_mem memobj)

Sampler objects
cl_sampler clCreateSamplerWithProperties (cl_context context, cl_int sampler_properties[] * sampler_properties, cl_int * errcode_ret)
c_l_int clGetSamplerInfo (cl_sampler sampler, cl_sampler_info param_name, size_t * param_value_size, const void * param_value, const cl_sampler_properties * pattern, cl_mem object, void * user_data, cl_int * errcode_ret)

Separate compilation and linking
cl_l_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, void *user_data), void *user_data, cl_int * errcode_ret)

Unload the OpenCL compiler
cl_l_int clUnloadPlatformCompiler (cl_platform_id platform)

Query program objects
cl_l_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)

Migrate memory objects
cl_l_int clEnqueueMigrateMemObjects (cl_command_queue command_queue, void *svm_mem_objects[], cl_uint num_mem_objects, cl_int mem_migration_flags, cl_uint num_events_in_wait_list, cl_event * event_wait_list, cl_event * event)

Flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query memory object
cl_l_int clGetMemObjectInfo (cl_mem memobj, cl_mem_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)

Sampler declaration fields [C 6.13.14]
The sampler can be passed as an argument to the kernel using cIsKernelArg, 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.

Sampler 
const sampler_t sampler = clCreateSamplerWithProperties (
    cl_context context, 
    sampler_properties[] sampler_properties, 
    errcode_ret
);

Separate compilation and linking
cl_l_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, void *user_data), void *user_data, cl_int * errcode_ret)

Unload the OpenCL compiler
cl_l_int clUnloadPlatformCompiler (cl_platform_id platform)

Query program objects
cl_l_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)

Migrate memory objects
cl_l_int clEnqueueMigrateMemObjects (cl_command_queue command_queue, void *svm_mem_objects[], cl_uint num_mem_objects, cl_int mem_migration_flags, cl_uint num_events_in_wait_list, cl_event * event_wait_list, cl_event * event)

Flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query memory object
cl_l_int clGetMemObjectInfo (cl_mem memobj, cl_mem_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)

Sampler declaration fields [C 6.13.14]
The sampler can be passed as an argument to the kernel using cIsKernelArg, 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.

Sampler 
const sampler_t sampler = clCreateSamplerWithProperties (
    cl_context context, 
    sampler_properties[] sampler_properties, 
    errcode_ret
);

Separate compilation and linking
cl_l_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, void *user_data), void *user_data, cl_int * errcode_ret)

Unload the OpenCL compiler
cl_l_int clUnloadPlatformCompiler (cl_platform_id platform)

Query program objects
cl_l_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)

Migrate memory objects
cl_l_int clEnqueueMigrateMemObjects (cl_command_queue command_queue, void *svm_mem_objects[], cl_uint num_mem_objects, cl_int mem_migration_flags, cl_uint num_events_in_wait_list, cl_event * event_wait_list, cl_event * event)

Flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query memory object
cl_l_int clGetMemObjectInfo (cl_mem memobj, cl_mem_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)

Sampler declaration fields [C 6.13.14]
The sampler can be passed as an argument to the kernel using cIsKernelArg, 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.

Sampler 
const sampler_t sampler = clCreateSamplerWithProperties (
    cl_context context, 
    sampler_properties[] sampler_properties, 
    errcode_ret
);

Separate compilation and linking
cl_l_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, void *user_data), void *user_data, cl_int * errcode_ret)

Unload the OpenCL compiler
cl_l_int clUnloadPlatformCompiler (cl_platform_id platform)

Query program objects
cl_l_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t * param_value_size, void *param_value, size_t * param_value_size_ret)
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_kernel clCreateKernelsInProgram (cl_program program, cl_uint num_kernels, cl_kernel *kernels, cl_uint *num_kernels_ret)
cl_kernel clRetainKernel (cl_kernel kernel)
cl_kernel clReleaseKernel (cl_kernel kernel)

Kernel arguments and queries
cl_int clSetKernelArg (cl_kernel kernel, cl_uint arg_index, size_t arg_size, void *arg_value)
cl_int clSetKernelArgSVMPointer (cl_kernel kernel, cl_uint arg_index, size_t arg_size, 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)

Execute kernels
cl_int clEnqueueNDRangeKernel (cl_command_queue command_queue, cl_kernel kernel, cl_uint work_dim, const size_t *global_work_size, const size_t *local_work_size, cl_uint num_events_in_wait_list, cl_event *eventWaitList, cl_event *event)

Markers, barriers, & waiting for events
cl_int clEnqueueMarkerWithWaitList (cl_command_queue command_queue, cl_uint num_events_in_wait_list, cl_event *eventWaitList)
cl_int clEnqueueBarrierWithWaitList (cl_command_queue command_queue, cl_uint num_events_in_wait_list, cl_event *eventWaitList)

Profiling operations
cl_int clGetEventProfilingInfo (cl_event event, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

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
event clCreateUserEvent (cl_context context, cl_int errcode_ret)
event clSetUserEventStatus (event event, cl_int execution_status)
event clWaitForEvents (cl_uint num_events, const cl_event *event_list)
event clGetEventInfo (cl_event event, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

Optimization options:
- cl-opt-disable
- cl-no-signed-zeros
- cl-uniform-work-group-size

Warning request/suppress:
- cl-uniform-work-group-size

Query kernel argument information:
- kernel-arg-info

Program Objects (continued)

Compiler options
Preprocessor:
- -D processed in order for clBuildProgram or clCompileProgram
- -D name=definition -I dir

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

Memory Model: SVM
A shared virtual memory 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

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

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-no-signed-zeros
- cl-finite-math-only
- cl-fast-relaxed-math
### Supported Data Types [C 6.1]
Double types require that `CLDEVICE ĐềQUE DOUBLE_FP_CONFIG` is not zero.

#### Built-in Scalar Data Types

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>--</td>
<td>true (1) or false (0)</td>
</tr>
<tr>
<td>char</td>
<td>cl_char</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>unsigned char, uchar</td>
<td>cl_uchar</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>short</td>
<td>cl_short</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>unsigned short, ushort</td>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>int</td>
<td>cl_int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>unsigned int, uint</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>unsigned long, ulong</td>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
</tr>
</tbody>
</table>

#### Built-in Vector Data Types

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>cl_float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>double</td>
<td>cl_double</td>
<td>64-bit IEEE 754</td>
</tr>
</tbody>
</table>

#### Vector Addressing Equivalences

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
<td>vl</td>
<td>vh</td>
</tr>
</tbody>
</table>

#### Preprocessor Directives & Macros [C 6.10]

- `__ENDIAN LITTLE__` : 1 if device is little endian
- `__IMAGE_SUPPORT__` : 1 if images are supported
- `__FAST RELAXED_MATH__` : 1 if `-O`fast-relaxed-math optimization option is specified
- `__kernel_exec(X, tpir)` : Same as: `__kernel __attribute__((vec_type_hint((work_group_size(X, Y, Z))) type defaults to int)) __attribute__((vec_type_hint((work_group_size(X, Y, Z)))) __attribute__(opencl_unroll_hints(Y, Z)))

### 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__((vec_type_hint((work_group_size(X, Y, Z))) __attribute__(opencl_unroll_hints(Y, Z)))` : type defaults to int
- `__attribute__((reqd_work_group_size(X, Y, Z)))`

#### Attribute Qualifiers [C 6.11]

- `__attribute__((aligned(n)))` : attribute[ يكن]host]
- `__attribute__((aligned))` : attribute[ يكن]device]
- `__attribute__((packed))` : attribute[ يكن]

### Conversions, Type Casting Examples [C 6.2]

- `T a = T(b);` : Scalar to scalar, or scalar to vector
- `T a = convert_T(b);` : Conversion to T
- `T a = as_T(b);` : Cast to T
- `T a = convert_T_sat(b);` : Cast to T, signed

### Built-in Vector Data Types

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float2</td>
<td>cl_float2</td>
<td>2D image array</td>
</tr>
<tr>
<td>double2</td>
<td>cl_double2</td>
<td>2D image array</td>
</tr>
</tbody>
</table>

### Operators

- `==` : equal to
- `!=` : not equal to
- `>` : greater than
- `<` : less than
- `>=` : greater than or equal to
- `<=` : less than or equal to
- `&` : and
- `|` : or
- `^` : exclusive or
- `<<` : left shift
- `>>` : right shift
- `?` : ternary operator
- `:` : assignment
- `=`, `op` : assignment
- `sizeof` : size of

### Preprocessor Directives & Macros

- `#pragma OPENCL_FP_CONTRACT` : on-off-switch: ON, OFF, DEFAULT
- `#pragma OPENCL EXTENSION extensionname : behavior`
- `#pragma OPENCL C_VERSION`: sub. integer for OpenCL C version
Work-Item Built-in Functions [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. Functions shown in blue require the feature macro _opencl_c_.

```
uint get_work_dim ()
size_t get_global_dim ()
size_t get_global_linear_id ()
size_t get_global_id ()
size_t get_local_dim ()
size_t get_local_linear_id ()
size_t get_local_id ()
size_t get_num_groups ()
size_t get_group_id ()
```

```
size_t get_global_offset ()
size_t get_global_linear_id ()
size_t get_local_linear_id ()
size_t get_sub_group_size ()
size_t get_max_sub_group_size ()
size_t get_num_sub_groups ()
size_t get_num_sub_groups ()
size_t get_global_linear_id ()
size_t get_group_id ()
size_t get_sub_group_linear_id ()
```

Math Constants [6.13.2]

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

```
#define MAXFLOAT value of maximum non-infinite single-precision floating-point number
#define HUGE_VALF positive float expression, evaluates to +infinity
#define HUGE_VAL positive double expression, evals. to +infinity (Requires double precision support.)
#define INFINITY constant float expression, positive or unsigned infinity
#define NANNAN 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.
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, float4) coord)
float4 read_imagef(image2d_array_t image, sampler_t sampler, (int2, float2) coord)
int4 read_imagei(image2d_array_t image, sampler_t sampler, (int2, float2) coord)
uint4 read_imageui(image2d_array_t image, sampler_t sampler, (int2, float4) coord)
float4 read_imagef(image2d_array_depth_t image, sampler_t sampler, (int2, float2) coord)

float4 write_imagef(image1d_t image, coord, float4 color)
int4 write_imagei(image1d_t image, coord, int4 color)
uint4 write_imageui(image1d_t image, coord, uint4 color)
float4 write_imagef(image1d_buffer_t image, coord, float4 color)
int4 write_imagei(image1d_buffer_t image, coord, int4 color)
uint4 write_imageui(image1d_buffer_t image, coord, uint4 color)
float4 write_imagef(image1d_array_t image, int2 coord, float4 color)
int4 write_imagei(image1d_array_t image, int2 coord, int4 color)
uint4 write_imageui(image1d_array_t image, int2 coord, uint4 color)
float4 write_imagef(image1d_array_depth_t image, int2 coord, float4 color)
int4 write_imagei(image1d_array_depth_t image, int2 coord, int4 color)
uint4 write_imageui(image1d_array_depth_t image, int2 coord, uint4 color)
float4 write_imagef(image3d_t image, coord, float4 color)
int4 write_imagei(image3d_t image, coord, int4 color)
uint4 write_imageui(image3d_t image, coord, uint4 color)
float4 write_imagef(image1d_buffer_t image, coord, float4 color)
int4 write_imagei(image1d_buffer_t image, coord, int4 color)
uint4 write_imageui(image1d_buffer_t image, coord, uint4 color)
float4 write_imagef(image1d_array_t image, int2 coord, float4 color)
int4 write_imagei(image1d_array_t image, int2 coord, int4 color)
uint4 write_imageui(image1d_array_t image, int2 coord, uint4 color)
float4 write_imagef(image1d_array_depth_t image, int2 coord, float4 color)
int4 write_imagei(image1d_array_depth_t image, int2 coord, int4 color)
uint4 write_imageui(image1d_array_depth_t image, int2 coord, uint4 color)
float4 write_imagef(image3d_t image, coord, float4 color)
int4 write_imagei(image3d_t image, coord, int4 color)
uint4 write_imageui(image3d_t image, coord, uint4 color)

Query image dimensions

int2 get_image_dim(image2d_t image)
int2 get_image_dim(image2d_array_t image)
int2 get_image_dim(image2d_array_depth_t image)

Query image channel data type and order

int get_image_channel_data_type(image2d_t image)
int get_image_channel_data_type(image2d_array_t image)
int get_image_channel_data_type(image2d_array_depth_t image)
int get_image_channel_data_type(image3d_t image)
int get_image_channel_data_type(image1d_t image)
int get_image_channel_data_type(image1d_buffer_t image)
int get_image_channel_data_type(image1d_array_t image)
int get_image_channel_data_type(image1d_array_depth_t image)
int get_image_channel_data_type(image3d_t image)
int get_image_channel_data_type(image1d_buffer_t image)
int get_image_channel_data_type(image1d_array_t image)
int get_image_channel_data_type(image1d_array_depth_t image)

Common Built-in Functions [C 6.13.14]
These functions operate component-wise and use round to nearest even rounding mode. Ts is type float, optionally double (if double precision is supported). Ts 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)
T degrees (T radians)
T mix (T x, T y, T a)
T step (T edge, T)
T smoothstep (T edge0, T edge1, T x)
T sign (T x)

©2020 Khronos Group - Rev. 0420
www.khronos.org/opencl
**Integer Built-in Functions** [C 6.13.3]

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

- \texttt{Tu abs} \((T x)\) \(\mid x \mid\)
- \texttt{Tu abs_diff} \((T x, T y)\) \(\mid x - y \mid\) without modulo overflow
- \texttt{Tu add_sat} \((T x, T y)\) \(x + y\) and saturates the result
- \texttt{Tu hadd} \((T x, T y)\) \((x + y) \gg 1\) without mod. overflow
- \texttt{Tu radd} \((T x, T y)\) \((x + y + 1) \gg 1\)
- \texttt{Tc clamp} \((T x, T y, T z)\) \(\min(min(x, y), z)\)
- \texttt{Tc clip} \((T x)\) \(0 \leq x < n\)
- \texttt{Tn mad_hi} \((T x, T y, T z)\) \(x y + z\)
- \texttt{Tn mad_sat} \((T a, T b, T c)\) \(a + b \cdot c\) and saturates the result
- \texttt{Tn max} \((T x, T y)\) \(\max(x, y)\)
- \texttt{Tn max} \((T x, T n y)\) \(\max(x, y)\) for \(n x\) or \(y\) otherwise it returns \(x\)
- \texttt{Tn min} \((T x, T y)\) \(\min(x, y)\)
- \texttt{Tn min} \((T n x, T y)\) \(\min(x, y)\) for \(n x\) or \(y\) otherwise it returns \(y\)
- \texttt{Tn mul_hi} \((T x, T y)\) High half of the product of \(x\) and \(y\)
- \texttt{Tn rotate} \((T x, T y)\) result\([\text{idx}] = v[\text{idx}] \ll \ll [\text{idx}]\)

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

- \texttt{Tn isequal} \((T x, T y)\) \(x = y\)
- \texttt{Tn isequal} \((T x, T y)\) \(x = y\)
- \texttt{Tn isnequal} \((T x, T y)\) \(x \neq y\)
- \texttt{Tn isnotequal} \((T x, T y)\) \(x \neq y\)
- \texttt{Tn isgreater} \((T x, T y)\) \(x > y\)
- \texttt{Tn isgreater} \((T x, T y)\) \(x > y\)
- \texttt{Tn isgreaterequal} \((T x, T y)\) \(x \geq y\)
- \texttt{Tn isgreaterequal} \((T x, T y)\) \(x \geq y\)
- \texttt{Tn isless} \((T x, T y)\) \(x < y\)
- \texttt{Tn isless} \((T x, T y)\) \(x < y\)
- \texttt{Tn islessgreater} \((T x, T y)\) \(x < y\) or \(x > y\)
- \texttt{Tn islessgreater} \((T x, T y)\) \(x < y\) or \(x > y\)
- \texttt{Tn islessor} \((T x, T y)\) \(x \leq y\)
- \texttt{Tn islessor} \((T x, T y)\) \(x \leq y\)
- \texttt{Tn isrelated} \((T x, T y)\) \((x \leq y) \lor (y \leq x)\)
- \texttt{Tn isrelated} \((T x, T y)\) \((x \leq y) \lor (y \leq x)\)

**Vector Data Load/Store** [C 6.13.7]

T is type char, uchar, short, ushort, int, uint, ulong, or float, optionally double (if double precision is supported). \(n\) refers to vector form of T, where \(n\) is 2, 3, 4, 8, or 16.

- \texttt{Tn vload} \((size\_t offset, const\_constant [constant] * p)\) Load vector from data address \((p + (offset * n))\)
- \texttt{Tn vstore} \((size\_t offset, const\_constant [constant] * p)\) Write vector to data address \((p + (offset * n))\)
- \texttt{Tn vloadl} \((size\_t offset, const\_constant [constant] * p)\) Load a half from address \((p + (offset * n))\)
- \texttt{Tn vstorel} \((size\_t offset, const\_constant [constant] * p)\) Write a half to address \((p + (offset * n))\)

**Geometric Built-in Functions** [C 6.13.5]

T is scalar type float, optionally double (if double precision is supported). T is the vector version of T.

- \texttt{float abs} \((\text{float} x)\) \(|x|\)
- \texttt{float abs} \((\text{double} x)\) \(|x|\)
- \texttt{float max} \((\text{float} x, \text{float} y)\) \(\max(x,y)\)
- \texttt{float max} \((\text{double} x, \text{double} y)\) \(\max(x,y)\)
- \texttt{float min} \((\text{float} x, \text{float} y)\) \(\min(x,y)\)
- \texttt{float min} \((\text{double} x, \text{double} y)\) \(\min(x,y)\)
- \texttt{float isgreatest} \((\text{float} x)\) \(x > 0\)
- \texttt{float isgreatest} \((\text{double} x)\) \(x > 0\)
- \texttt{float isgreatest} \((\text{float} x, \text{float} y)\) \(\max(x,y) > 0\)
- \texttt{float isgreatest} \((\text{double} x, \text{double} y)\) \(\max(x,y) > 0\)
- \texttt{float isgreatestequal} \((\text{float} x, \text{float} y)\) \(x = y > 0\)
- \texttt{float isgreatestequal} \((\text{double} x, \text{double} y)\) \(x = y > 0\)

- \texttt{float fast_distance} \((\text{float} p0, \text{float} p1)\) \(\sqrt{(p_0 - p_1)^2}\)
- \texttt{float fast_distance} \((\text{double} p0, \text{double} p1)\) \(\sqrt{(p_0 - p_1)^2}\)
- \texttt{float fast_distance} \((\text{float} p0, \text{float} p1)\) \(\sqrt{(p_0 - p_1)^2}\)
- \texttt{float fast_distance} \((\text{double} p0, \text{double} p1)\) \(\sqrt{(p_0 - p_1)^2}\)

- \texttt{float fast_normalize} \((\text{float} p0)\) \(\frac{p0}{\sqrt{p0^2 + p0^2 + p0^2 + p0^2}}\)
- \texttt{float fast_normalize} \((\text{double} p0)\) \(\frac{p0}{\sqrt{p0^2 + p0^2 + p0^2 + p0^2}}\)

- \texttt{float isless} \((\text{float} x, \text{float} y)\) \(x < y\)
- \texttt{float isless} \((\text{double} x, \text{double} y)\) \(x < y\)
- \texttt{float islessor} \((\text{float} x, \text{float} y)\) \(x \leq y\)
- \texttt{float islessor} \((\text{double} x, \text{double} y)\) \(x \leq y\)
- \texttt{float islessgreatest} \((\text{float} x, \text{float} y)\) \(x > y \text{ and saturates the result}\)
- \texttt{float islessgreatest} \((\text{double} x, \text{double} y)\) \(x > y \text{ and saturates the result}\)

- \texttt{float fast_distance} \((\text{float} p0, \text{float} p1)\) \(\sqrt{(p_0 - p_1)^2}\)
- \texttt{float fast_distance} \((\text{double} p0, \text{double} p1)\) \(\sqrt{(p_0 - p_1)^2}\)
- \texttt{float fast_distance} \((\text{float} p0, \text{float} p1)\) \(\sqrt{(p_0 - p_1)^2}\)
- \texttt{float fast_distance} \((\text{double} p0, \text{double} p1)\) \(\sqrt{(p_0 - p_1)^2}\)

- \texttt{float fast_normalize} \((\text{float} p0)\) \(\frac{p0}{\sqrt{p0^2 + p0^2 + p0^2 + p0^2}}\)
- \texttt{float fast_normalize} \((\text{double} p0)\) \(\frac{p0}{\sqrt{p0^2 + p0^2 + p0^2 + p0^2}}\)
### Synchronization & Memory Fence Functions [C 6.13.8]

flags argument is the memory address space, set to a D 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])`
- `void sub_group_barrier(cl_mem_fence_flags flags [, memory_scope scope])`

### Miscellaneou Vector Functions [C 6.13.12]

Tm and Tn are type char, uchar, short, ushort, int, intra, uint, long, ulong, float, or double.

- `void shuffle(Tm x, Tn y)`: Constructs 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.
- `int vec_step ( Tn a )`: 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.

### Atomic Functions [C 6.13.11]

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

- `void atomic_init(volatile A *obj, C value)` initializes the atomic object pointed to by obj to the value value.
- `void atomic_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 *obj, C value, memory_order order [, memory_scope scope])`
- `atomic_store_explicit(volatile A *obj, C value, memory_order order [, memory_scope scope])`

### Address Space Qualifier Functions [C 6.13.9]

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

- `bool atomic_flag_test_and_set(volatile atomic_flag *obj)` atomically sets the value pointed to by obj 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 *obj)` atomically sets the value pointed to by obj to false. The order argument shall not be memory_order_acquire or memory_order_acq_rel.

### Values for key for atomic_fetch and modify functions

<table>
<thead>
<tr>
<th>key</th>
<th>op</th>
<th>computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>op</td>
<td>computation</td>
</tr>
<tr>
<td>add</td>
<td>+</td>
<td>bitwise and</td>
</tr>
<tr>
<td>sub</td>
<td>-</td>
<td>min</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td>max</td>
</tr>
</tbody>
</table>

### Atomic Types and Enum Constants

**Parameter type: memory_order**

<table>
<thead>
<tr>
<th>Values</th>
<th>Optional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_order_acq_rel</td>
<td>Requires OpenCL C 2.0 or __opencl_c_atomic_order_seq_cst</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Values</th>
<th>Optional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_scope_work_group</td>
<td>Only used with atomic_work_item_fence with flags: CLK_IMAGE_MEM_FENCE</td>
</tr>
<tr>
<td>memory_scope_all_svm_devices</td>
<td>Requires __opencl_c_all_svm_devices</td>
</tr>
</tbody>
</table>

(Continued on next page)
Atomic Functions (continued)

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

<table>
<thead>
<tr>
<th>atomic_int</th>
<th>atomic_double</th>
<th>atomic_ptrdiff_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic_uint</td>
<td>atomic_long</td>
<td>atomic_intptr_t</td>
</tr>
<tr>
<td>atomic_flag</td>
<td>atomic_ulong</td>
<td>atomic_uintptr_t</td>
</tr>
<tr>
<td>atomic_float</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legacy Atomic Functions

These functions provide 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 or global volatile __local__.

Pipe Built-in Functions

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

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__opencl_c_pipes</td>
<td>Provides support for atomic built-in functions.</td>
</tr>
<tr>
<td>__opencl_c_work_group_collective_functions</td>
<td>Supports collective functions such as barrier and flush within a work-group.</td>
</tr>
<tr>
<td>__opencl_c_work_group_reduction_functions</td>
<td>Supports reduction operations within a work-group.</td>
</tr>
<tr>
<td>__opencl_c_work_group_scan_functions</td>
<td>Supports scan operations within a work-group.</td>
</tr>
<tr>
<td>__opencl_c_work_group_scan_exclusive_functions</td>
<td>Supports exclusive scan operations within a work-group.</td>
</tr>
<tr>
<td>__opencl_c_work_group_broadcast</td>
<td>Broadcasts a value to all work-items in the work-group.</td>
</tr>
<tr>
<td>__opencl_c_work_group_fence</td>
<td>Signals completion of all work-items in the work-group.</td>
</tr>
<tr>
<td>__opencl_c_work_group_commit_read_pipe</td>
<td>Commits read operations to a pipe.</td>
</tr>
<tr>
<td>__opencl_c_work_group_commit_write_pipe</td>
<td>Commits write operations to a pipe.</td>
</tr>
<tr>
<td>__opencl_c_work_group_reserve_read_pipe</td>
<td>Reserves area for reading from a pipe.</td>
</tr>
<tr>
<td>__opencl_c_work_group_reserve_write_pipe</td>
<td>Reserves area for writing to a pipe.</td>
</tr>
<tr>
<td>__opencl_c_work_group_reserve_read_area</td>
<td>Reserves area for reading from a pipe.</td>
</tr>
<tr>
<td>__opencl_c_work_group_reserve_write_area</td>
<td>Reserves area for writing to a pipe.</td>
</tr>
</tbody>
</table>

Notes

- The above two print calls print the following:
  - for printf: `f4 = 1.002, 0.030, 4.00`  
  - for wprintf: `uc = 0x0f, 0x0b, 0x0c, 0x0d`

- Examples show the use of the vector specifier in the printf format string.
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_event 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_subgroups_nesting__. All other functions require support for __opencl_c_device_enqueue or OpenCL C 2.0.

```c
int enqueue kernel (queue_t queue, kernel enqueue flags, const ndrange_t ndrange, void (*block)(void))
int enqueue kernel (queue_t queue, kernel enqueue flags, const ndrange_t ndrange, num_events_in_wait_list, const ck_event_t *event_wait_list, ck_event_t *event_ret, void (*block)(void))
int enqueue kernel (queue_t queue, kernel enqueue flags, const ndrange_t ndrange, void (*block)(void), …, uint sized, …)
int enqueue kernel (queue_t queue, kernel enqueue flags, const ndrange_t ndrange, num_events_in_wait_list, const ck_event_t *event_wait_list, ck_event_t *event_ret, void (*block)(void), …, uint sized, …)
```

Enqueue 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 __OPENCL_ENUMFLAGS__.

```c
void (*block)(void) = NULL
```

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.

```c
void retain event (ck_event_t event) Increment event reference count.
void release event (ck_event_t event) Decrement event reference count.
bool is valid event (ck_event_t event) True for valid event.
void set_user event status (ck_event_t event, int status) Sets the execution status of a user event.
```

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.

```c
queue_t get default queue (void) Default queue or CLK_NULL_QUEUE.
nrange_t ndrange_1D (size_t global work size), size_t local work size) Builds a 1D ND-range descriptor.
nrange_t ndrange_1D (size_t global work offset, size_t global work size, size_t local work size) Builds a 2D or 3D ND-range descriptor. n may be 2 or 3.
nrange_t ndrange_nD (const size_t global work size[], const size_t local work size[]) Builds an ND-range descriptor.
nrange_t ndrange_nD (const size_t global work offset, const size_t global work size, const size_t local work size[]) Builds an ND-range descriptor.
```

Feature Macros [C Appendix A]

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

<table>
<thead>
<tr>
<th>Feature Macro</th>
<th>The OpenCL C Compiler Supports...</th>
</tr>
</thead>
<tbody>
<tr>
<td>__opencl_c_3d_image writes</td>
<td>Built-in functions for writing to 3D image objects.</td>
</tr>
<tr>
<td>__opencl_c_atomic_order_acq_rel</td>
<td>Enumerations and built-in functions for atomic operations with acquire and release memory consistency orders.</td>
</tr>
<tr>
<td>__opencl_c_atomic_order_seq_cst</td>
<td>Enumerations and built-in functions for atomic operations and fences with sequentially consistent memory consistency order.</td>
</tr>
<tr>
<td>__opencl_c_atomic_scope_device</td>
<td>Enumerations and built-in functions for atomic operations and fences with device memory scope.</td>
</tr>
<tr>
<td>__opencl_c_atomic_scope_all_svm_devices</td>
<td>Enumerations and built-in functions for atomic operations and fences with all SVM devices memory scope.</td>
</tr>
<tr>
<td>__opencl_c_device_enqueue</td>
<td>Built-in functions to enqueue additional work from the device.</td>
</tr>
<tr>
<td>__opencl_c_GENERIC_ADDRESS_SPACE</td>
<td>The unnamed generic address space.</td>
</tr>
<tr>
<td>__opencl_c_Pipes</td>
<td>The pipe modifier and built-in functions to read and write from a pipe.</td>
</tr>
<tr>
<td>__opencl_c_program_scope_global_variables</td>
<td>Program scope variables in the global address space.</td>
</tr>
<tr>
<td>__opencl_c_read_write_images</td>
<td>Reading from and writing to the same image object in a kernel.</td>
</tr>
<tr>
<td>__opencl_c_subgroups</td>
<td>Built-in functions operating on sub-groupings of work-items.</td>
</tr>
<tr>
<td>__opencl_c_workgroup_collective_functions</td>
<td>Built-in functions that perform collective operations across a work-group.</td>
</tr>
</tbody>
</table>

OpenCL Device Architecture Diagram

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

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.

OpenCL is a trademark of Apple Inc. and is used under license by Khronos.