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

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_id, cl_platform_info param_name, size_t * param_value_size, void * param_value, size_t * param_value_size_ret)

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

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)

cl_int clGetDeviceAndHostTimer (cl_device_id device, size_t * param_value_size, void * param_value, size_t * param_value_size_ret)

**Partitioning a device [API 4.3]**

cl_int clCreateSubDevices (cl_device_id in_device, cl_device_id * out_devices, cl_uint numDevices)

cl_int clGetDeviceAndHostTimer (cl_device_id device, cl_ulong * timestamp)

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, cl_int * errcode_ret)

*properties*: NULL or a pointer to a non-terminating list of properties and their values:

- **CL_QUEUE_PROPERTIES** (bitfield which may be set to an OR of CL_QUEUE_ * where * may be:
  - **OUT_OF_ORDER_EXEC_MODE_ENABLE**, **ON_DEVICE_DEFAULT**), **PROILING_ENABLE**

The OpenCL API Reference

OpenCL 3.0 Reference Guide

OpenCL 3.0 API specification

OpenCL 3.0 C Language specification

OpenCL 3.0 Extension specification
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_int 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_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

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, 
  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 *offset, size_t *size, size_t *region, void *ptr, cl_uint num_events_in_wait_list, 
  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 *offset, size_t *size, size_t *region, void *ptr, cl_uint num_events_in_wait_list, 
  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, 
  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 *offset, size_t *size, size_t *region, void *ptr, cl_uint num_events_in_wait_list, 
  cl_event *event_wait_list, cl_event *event)

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

map_flags: CL_MAP_READ, WRITE, CL_MAP_WRITE_INVALIDATE_REGION

Image Formats [API 5.3.1]

Create image objects

cl_mem clCreateImage ( 
  cl_context context, cl_mem_properties *properties, cl_mem_flags *flags, 
  const cl_image_format *image_format, cl_int *errcode_ret)
flags: See clCreateImage

cl_mem clCreateImageWithProperties ( 
  cl_context context, const cl_mem_properties *properties, cl_mem_flags *flags, 
  const cl_image_format *image_format, cl_int *errcode_ret)
flags: See clCreateImage

Query list of supported image formats

cl_int clGetSupportedImageFormats ( 
  cl_context context, cl_mem_properties *properties, cl_mem_object_type image_type, 
  cl_uint num_entries, cl_image_format *image_formats, 
  cl_int *errcode_ret)
flags: See clCreateImage

Image object parameters

cl_int clGetImageInfo ( 
  cl_image image, cl_image_info param_name, cl_uint *param_value_size, 
  void *param_value, cl_int *param_value_size_ret)
param_name: CL_IMAGE_FORMAT, CL_IMAGE_ARRAY_ELEMENT_SIZE, 
CL_IMAGE_N, CL_IMAGE_ARRAY_MAX_SIZE, CL_IMAGE_ARRAY_HEIGHT, CL_IMAGE_ARRAY_WIDTH, 
CL_IMAGE_ARRAY_DEPTH, CL_IMAGE_ARRAY_DATA_SIZE, CL_IMAGE_ARRAY_DATA_TYPE, 
CL_IMAGE_ARRAY_NAME, CL_IMAGE_ARRAY_MIP_LEVELS)

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

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

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

cl_int clEnqueueImageCopy ( 
  cl_command_queue command_queue, cl_mem source_image, cl_mem dst_image, 
  const size_t *src_offset, size_t *dst_offset, size_t *size, 
  const size_t *region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueImageCopyToBuffer ( 
  cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer, 
  const size_t *src_offset, size_t *dst_offset, size_t *size, 
  const size_t *region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueImageCopyToBufferRect ( 
  cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer, 
  const size_t *src_offset, size_t *dst_offset, size_t *size, 
  const size_t *region, cl_uint num_events_in_wait_list, 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 void *pattern, 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_N, CL_IMAGE_ARRAY_MAX_SIZE, CL_IMAGE_ARRAY_HEIGHT, CL_IMAGE_ARRAY_WIDTH, 
CL_IMAGE_ARRAY_DEPTH, CL_IMAGE_ARRAY_DATA_SIZE, CL_IMAGE_ARRAY_DATA_TYPE, 
CL_IMAGE_ARRAY_NAME, CL_IMAGE_ARRAY_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, cl_int pipe_max_packets, const cl_pipe_properties *properties, cl_int *encode_retry);
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_t param_value_size, void *param_value, size_t_t *param_value_size_ret);

Pipes (Continued on next page >)

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_memmem flags, size_t_t size, cl_uint alignment);
flags: CL_MEM_READ_WRITE, CL_MEM_WRITE_READ, CL_MEM_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 *svm_pointers[], void (CL_CALLBACK*fn_free_func)(cl_mem mem);
cl_int clEnqueueSVMMap (cl_command_queue command_queue, cl_bool blocking_copy, void *dst_ptr, const void *src_ptr, size_t_t size, cl_uint num_events_in_wait_list, cl_int *event_wait_list, cl_event *event);
cl_int clEnqueueSVMMemcp (cl_command_queue command_queue, cl_bool blocking_copy, void *dst_ptr, const void *src_ptr, size_t_t size, cl_uint num_events_in_wait_list, cl_int *event_wait_list, cl_event *event);
cl_int clEnqueueSVMMemfill (cl_command_queue command_queue, const void *pattern, size_t_t size, cl_uint num_events_in_wait_list, cl_int *event_wait_list, cl_event *event);
cl_int clEnqueueSVMMap (cl_command_queue command_queue, cl_bool blocking_copy, cl_memflags map_flags map_flags, void *svm_ptr, size_t_t size, cl_uint num_events_in_wait_list, cl_int *event_wait_list, cl_event *event);
cl_int clEnqueueSVMMemUnmap (cl_command_queue command_queue, void *svm_ptr, cl_uint num_events_in_wait_list, cl_int *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, cl_int *event_wait_list, cl_event *event);

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 cl RetrieveMemObject (cl_mem memobj, cl_int *memobj);
cl_int cl ReleaseMemObject (cl_mem memobj, cl_int *memobj);
cl_int clSetMemObjectD estructorCallback (cl_mem memobj, void (CL_CALLBACK*fn_profit_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, cl_int *event_wait_list, cl_event *event);

Sampler Objects [API 5.7]
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.

Sampler declaration fields
cl_int clGetSamplerInfo (cl_sampler sampler, cl_sampler_info param_name, size_t_t param_value_size, void *param_value, size_t_t *param_value_size_ret);

Sampler declaration fields (Continued on next page >)

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*fn_profit_notify)(cl_mem memobj, void *mapped_ptr, cl_uint num_events_in_wait_list, cl_int *event_wait_list, cl_event *event);
cl_int 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*fn_profit_notify)(cl_program program, void *user_data, cl_int *event_wait_list, cl_event *event);
cl_int clReleaseProgram (cl_program program);

Unload the OpenCL compiler
cl_int clUnloadPlatformCompiler (cl_platform_id platform_id);

Query program objects
cl_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t_t param_value_size, void *param_value, size_t_t *param_value_size_ret);

cl_int clGetProgramBuildInfo (cl_program program, cl_program_build_info param_name, cl_int *binary_status, cl_int *cross_device_binary_status);

Program Objects [API 5.8]
OpenCL programs consist of sets of kernels identified as functions declared with the __kernel qualifier in the program source.

Program Objects (Continued on next page >)

Flush and Finish
cl_int clFlush (cl_command_queue command_queue);
cl_int clFinish (cl_command_queue command_queue);
Program Objects (continued)

Compiler options

Preprocessor:
- (D processed in order for cBuildProgram or ccl CompileProgram)
- (D name=definition -I dir

Math intrinsics:
- -D single-precision-constant
- -D define-are-zero
- -D fp32-correctly-rounded-divide-sqrt

OpenCL 3.0 Reference Guide

Explicit updates between host and device?

OpenCL Memory objects (buffer)
- Host memory allocation
- clSVMAlloc

Mechanisms to enforce consistency

OpenCL Memory objects (buffer)
- Yes, through Map and Unmap commands.

Granularity of sharing

OpenCL Memory objects (buffer)
- cl_event

Warning request/suppress:

OpenCL Memory objects (buffer)
- cl_int

clSVMAlloc

Mechanisms to enforce consistency

OpenCL Memory objects (buffer)
- cl_int

Yes, through Map and Unmap commands.

Synchronization points plus atomics (if

cl_event

Control OpenCL language version:

OpenCL Memory objects (buffer)
- cl_int

clSVMAlloc

Mechanisms to enforce consistency

OpenCL Memory objects (buffer)
- cl_int

Yes, through Map and Unmap commands.

Synchronization points plus atomics (if

cl_event

Query kernel argument information:

cl_int

clGetKernelArgInfo

cl_int

clGetKernelArgSVMPointer

cl_int

clSetKernelArg

cl_int

clReleaseKernel

Event Objects [API 5.9 - 5.10]

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

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

Summary of SVM options in OpenCL

SVM
- 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.

<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</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</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</td>
<td>clSVMAlloc</td>
<td>Synchronization points plus atoms (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 atoms (if supported)</td>
<td>No</td>
</tr>
</tbody>
</table>
Supported Data Types

**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>cl_int</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</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</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</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>64-bit signed. Support for the __opencl_c_int64 feature required.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>cl_ulong</td>
<td>64-bit unsigned. Support for the __opencl_c_int64 feature required.</td>
</tr>
<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. Support for the __opencl_c_double64 feature required.</td>
</tr>
<tr>
<td>half</td>
<td>cl_half</td>
<td>16-bit float (storage only)</td>
</tr>
<tr>
<td>size_t</td>
<td>cl_size_t</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>ptdiff_t</td>
<td>cl_ptdiff_t</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>nptr_t</td>
<td>cl_nptr_t</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>uintptr_t</td>
<td>cl_ulongt</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>void</td>
<td>void</td>
<td>void</td>
</tr>
</tbody>
</table>

**Built-in Vector Data Types**

The following types shown below require support for the __opencl_c_image feature.

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float^n</td>
<td>cl_floatn</td>
<td>32-bit float</td>
</tr>
<tr>
<td>double^n</td>
<td>cl_doublen</td>
<td>64-bit float. Support for the __opencl_c_double64 feature required.</td>
</tr>
<tr>
<td>half^n</td>
<td>cl_halfn</td>
<td>16-bit float. Support for the __opencl_c_half feature required.</td>
</tr>
<tr>
<td>size_t^n</td>
<td>cl_size_tn</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>ptdiff_t^n</td>
<td>cl_ptdiff_tn</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>nptr_t^n</td>
<td>cl_nptr_tn</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>uintptr_t^n</td>
<td>cl_ulongtn</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>void^n</td>
<td>voidn</td>
<td>voidn</td>
</tr>
</tbody>
</table>

**Reserved Data Types**

- quad, quadn: complex half, complex halfn
- intn: imaginary half, imaginary halfn
- floatn: complex float, complex floatn
- doublen: imaginary float, imaginary floatn
- longn: complex double, complex doublen
- longlongn: imaginary double, imaginary doublen
- longlonglongn: complex quad, complex quadn
- longlonglonglongn: imaginary quad, imaginary quadn
- floatnmx: long double, long doublen
- __opencl_c_int64
- __opencl_c_images
- __opencl_c_pipes
- __opencl_c_int64
- __opencl_c_pipes
- __opencl_c_image
- __opencl_c_int64

**Operators and Qualifiers**

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

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

**Address Space Qualifiers**

- __global, __local, __constant, __private, __private

**Function Qualifiers**

- __kernel, __kernel
- __attribute__((ivec_type_hint(type)))
- __attribute__((vec_type_hint(type)))
- __attribute__((vec_type_hint(type)))
- __attribute__((vec_type_hint(type)))

**Attribute Qualifiers**

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

- __alignof__(type)
- __attribute__((aligned)(alignment))
- __attribute__((aligned)(alignment))
- __attribute__((aligned)(alignment))
- __attribute__((aligned)(alignment))

Use to specify special attributes of variables or structure fields.

- __attribute__((aligned)(alignment))
- __attribute__((aligned)(alignment))
- __attribute__((aligned)(alignment))

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

- __attribute__((d1))
- __attribute__((d2))

**Conversions, Type Casting Examples**

- `T` = `T_n`; // Scalar to scalar, or scalar to vector
- `T_a` = `convert_T_b(T_b)`;
- `T_a` = `convert_T_f(T_f)`;
- `T_a` = `as_T_b(T_b)`;
- `T_a` = `T_b(T_b)`;
- `T_a` = `T_b(T_b)`;
- `R` = one of the rounding modes
  - nearest even
  - toward zero
  - toward + infinity
  - toward - infinity

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

**Preprocessor Directives & Macros**

- #pragma OPENCL_FP_CONTRACT on-off-switch: ON, OFF, DEFAULT
- #pragma OPENCL_EXTENSION : behavior
- #pragma OPENCL_EXTENSION all : behavior

- __FILE__
- __LINE__
- __OPENCL_VERSION__
- __CL_VERSION_1_0__
- __CL_VERSION_1_1__
- __CL_VERSION_1_2__
- __CL_VERSION_2_0__
- __CL_VERSION_3_0__
- __OPENCL_C_VERSION__
- __ENDIAN_LITTLE__

- __IMAGE_SUPPORT__
- __FAST_RELAXED_MATH__
- __OPENCL_C_VERSION__
- __CL_VERSION_3_0__
- __CL_VERSION_1_2__
- __CL_VERSION_1_1__
- __CL_VERSION_1_0__
- __OPENCL_C_VERSION__
- __ENDIAN_LITTLE__

©2022 Khronos Group - Rev. 0922
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.

```c
#pragma read_only
#pragma write_only
#pragma read_write
```

(Requires OpenCL C 2.0 or the __opencl_c_device_enqueue_image feature.)

### Blocks [C 6.12]

A result value type with a list of parameter types.

Requires support for the __opencl_c_device_enqueue feature 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 the environment.

```
int (*myBlock)(int) =
^ {return num * multiplier;};
```

---

### 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.

`T` is type float. If supported, `T` can also be type double.`Tn` is the vector form of `T`, where `n` is 2, 3, 4, 8, or 16. `T` is `T` 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.

**HN** indicates that half and native variants are available using only the float or float 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 types.

### Math Constants [C 6.13.3]

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
- **INFINITY** Constant float expression, positive or unsigned infinity
- **NAN** Constant float expression, quiet NaN

The values of the following symbolic constant is double-precision float:

- **HUGE_VAL** Positive double expression, evals. to +infinity (Requires double precision support.)

When double precision is supported, macros ending in _F from the macro name.

```
M_E_F  Value of e
M_LOG2E_F  Value of log_e
M_LOG10E_F  Value of log_e
M_LOG2_F  Value of log_2
M_LOG10_F  Value of log_10
M_PI_F  Value of pi
M_PI_2_F  Value of pi/2
M_PI_4_F  Value of pi/4
M_1_PI_F  Value of 1 / pi
M_2_PI_F  Value of 2 / pi
M_SQRT2_F  Value of sqrt(2)
M_SQRT1_2_F  Value of 1 / sqrt(2)
```

---

![OpenCL 3.0 Reference Guide](https://example.com/opencl_3.0_reference_guide.png)

OpenCL C Language Page 6

![OpenCL C Language](https://example.com/opencl_c_language.png)

© 2022 Khronos Group - Rev. 0922

www.khronos.org/opencl
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.
Qualifier 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 coord)
int4 read_imagei (read_only image2d_t image, sampler_t sampler, int2 coord)
uint4 read_imageui (read_only image2d_t image, sampler_t sampler, int2 coord)
float4 read_imagef (read_only image2d_array_t image, sampler_t sampler, int4 coord)
int4 read_imagei (read_only image2d_array_t image, sampler_t sampler, int4 coord)
uint4 read_imageui (read_only image2d_array_t image, sampler_t sampler, int4 coord)
float4 read_imagef (read_only image2d_array_depth_t image, sampler_t sampler, int2 coord, float depth)
int4 read_imagei (read_only image2d_array_depth_t image, sampler_t sampler, int2 coord, float depth)
uint4 read_imageui (read_only image2d_array_depth_t image, sampler_t sampler, int2 coord, float depth)
float4 read_imagef (aQual image2d_t image, int2 coord, int4 color)
int4 read_imagei (aQual image2d_t image, int2 coord, int4 color)
uint4 read_imageui (aQual image2d_t image, int2 coord, int4 color)
float4 read_imagef (aQual image2d_t image, int2 coord, uint4 color)
int4 read_imagei (aQual image2d_t image, int2 coord, uint4 color)
uint4 read_imageui (aQual image2d_t image, int2 coord, uint4 color)
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)
float4 read_imagef (aQual image2d_array_depth_t image, int2 coord, float depth)
int4 read_imagei (aQual image2d_array_depth_t image, int2 coord, float depth)
uint4 read_imageui (aQual image2d_array_depth_t image, int2 coord, float depth)
float4 read_imagef (image1d_t image, int coord, int4 color)
int4 read_imagei (image1d_t image, int coord, int4 color)
uint4 read_imageui (image1d_t image, int coord, int4 color)
float4 read_imagef (image1d_buffer_t image, int coord, int4 color)
int4 read_imagei (image1d_buffer_t image, int coord, int4 color)
uint4 read_imageui (image1d_buffer_t image, int coord, int4 color)
float4 read_imagef (image1d_array_t image, int2 coord, int4 color)
int4 read_imagei (image1d_array_t image, int2 coord, int4 color)
uint4 read_imageui (image1d_array_t image, int2 coord, int4 color)
float4 read_imagef (image1d_array_depth_t image, int2 coord, int4 color)
int4 read_imagei (image1d_array_depth_t image, int2 coord, int4 color)
uint4 read_imageui (image1d_array_depth_t image, int2 coord, int4 color)

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 coord)
int4 read_imagei (read_only image1d_t image, sampler_t sampler, int coord)
uint4 read_imageui (read_only image1d_t image, sampler_t sampler, int coord)
float4 read_imagef (read_only image1d_array_t image, sampler_t sampler, int2 coord, float depth)
int4 read_imagei (read_only image1d_array_t image, sampler_t sampler, int2 coord, float depth)
uint4 read_imageui (read_only image1d_array_t image, sampler_t sampler, int2 coord, float depth)
float4 read_imagef (aQual image1d_t image, int coord, int4 color)
int4 read_imagei (aQual image1d_t image, int coord, int4 color)
uint4 read_imageui (aQual image1d_t image, int coord, int4 color)
float4 read_imagef (aQual image1d_buffer_t image, int coord, int4 color)
int4 read_imagei (aQual image1d_buffer_t image, int coord, int4 color)
uint4 read_imageui (aQual image1d_buffer_t image, int coord, int4 color)
float4 read_imagef (aQual image1d_array_t image, int2 coord, int4 color)
int4 read_imagei (aQual image1d_array_t image, int2 coord, int4 color)
uint4 read_imageui (aQual image1d_array_t image, int2 coord, int4 color)
float4 read_imagef (image1d_t image, int coord, int4 color)
int4 read_imagei (image1d_t image, int coord, int4 color)
uint4 read_imageui (image1d_t image, int coord, int4 color)
float4 read_imagef (image1d_buffer_t image, int coord, int4 color)
int4 read_imagei (image1d_buffer_t image, int coord, int4 color)
uint4 read_imageui (image1d_buffer_t image, int coord, int4 color)
float4 read_imagef (image1d_array_t image, int2 coord, int4 color)
int4 read_imagei (image1d_array_t image, int2 coord, int4 color)
uint4 read_imageui (image1d_array_t image, int2 coord, int4 color)
float4 read_imagef (image1d_array_depth_t image, int2 coord, int4 color)
int4 read_imagei (image1d_array_depth_t image, int2 coord, int4 color)
uint4 read_imageui (image1d_array_depth_t image, int2 coord, int4 color)

Query image functions
Get image dimensions

int get_image_width (image2d_t image)
int get_image_width (image2d_array_t image)
int get_image_width (image2d_array_depth_t image)
int get_image_height (image2d_t image)
int get_image_height (image2d_array_t image)
int get_image_height (image2d_array_depth_t image)
int get_image_depth (image3d_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 (image2d_array_t image)
int get_image_channel_data_type (image2d_array_depth_t image)
int get_image_channel_order (image2d_t image)
int get_image_channel_order (image2d_array_t image)
int get_image_channel_order (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. T is type float. If supported, T can also be type double. Tn is the vector of Ts, where n is 2, 3, 4, 8, or 16. T is Ts and Tn.

float clamp (float x, float min, float max)
float clamp (float x, float min, float max)
degrees (float radians)
max (float x, float y)
max (float x, float y)
min (float x, float y)
min (float x, float y)
mix (float x, float y, float a)
mix (float x, float y, float a)
linearblend (float x, float y, float a)
linearblend (float x, float y, float a)
tdegrees (float radians)
tdegrees (float radians)
tstep (float edge, float)
tstep (float x, float y)
tsmoothstep (float edge0, float edge1, float x)
tsmoothstep (float x, float edge0, float edge1, float x)
tsign (float x)
tsign (float x)
### Integer Built-in Functions [C 6.13.3]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(T x)</td>
<td>Absolute value of x</td>
<td>x</td>
</tr>
<tr>
<td>abs_diff(T x, T y)</td>
<td>Absolute difference of x and y without modulo overflow</td>
<td></td>
</tr>
<tr>
<td>add_sat(T x, T y)</td>
<td>Sum of x and y saturates the result</td>
<td>x + y</td>
</tr>
<tr>
<td>add_clamp(T x, T y, T max)</td>
<td>Clip the result of x + y to Tmax</td>
<td>min(max(x, y), Tmax)</td>
</tr>
<tr>
<td>add_clamp(T x, T y, T min)</td>
<td>Clip the result of x + y to Tmin</td>
<td>max(min(x, y), Tmin)</td>
</tr>
<tr>
<td>clz(T x)</td>
<td>Number of leading 0-bits in x</td>
<td>n clz x</td>
</tr>
<tr>
<td>popcnt(T x)</td>
<td>Number of non-zero bits in x</td>
<td>n popcnt x</td>
</tr>
<tr>
<td>mul_hi(T x, T y)</td>
<td>Multiply x and y, return type is scalar when the parameters are scalar.</td>
<td>x * y</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>vread(T o, data, size_t offset, const [constant] T p)</td>
<td>Read vector data from address(p + offset * n)</td>
<td></td>
</tr>
<tr>
<td>vwrite(T o, data, size_t offset, const [constant] T p)</td>
<td>Write vector data to address(p + offset * n)</td>
<td></td>
</tr>
<tr>
<td>void vload(T o, size_t offset, T p)</td>
<td>Read a half from address(p + offset * n)</td>
<td></td>
</tr>
<tr>
<td>float vload_hi(T o, size_t offset, T p)</td>
<td>Read a half from address(p + offset * n)</td>
<td></td>
</tr>
<tr>
<td>float vload_half(T o, size_t offset, const [constant] T p)</td>
<td>Read a half from address(p + offset * n)</td>
<td></td>
</tr>
<tr>
<td>float vload_half(T o, size_t offset, const [constant] T p)</td>
<td>Read a half from address(p + offset * n)</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>isless(x, y)</td>
<td>Test if x is less than y</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>isless_equal(x, y)</td>
<td>Test if x is less than or equal to y</td>
<td>x &lt;= y</td>
</tr>
<tr>
<td>isgreater(x, y)</td>
<td>Test if x is greater than y</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>isgreater_equal(x, y)</td>
<td>Test if x is greater than or equal to y</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>isfinite(x)</td>
<td>Test if x is finite</td>
<td>x finite</td>
</tr>
<tr>
<td>isnormal(x)</td>
<td>Test if x is normal</td>
<td>x normal</td>
</tr>
<tr>
<td>isnan(x)</td>
<td>Test if x is NaN</td>
<td>x NaN</td>
</tr>
<tr>
<td>isnan(T o, data, size_t offset, const [constant] T p)</td>
<td>Test for a NaN</td>
<td></td>
</tr>
</tbody>
</table>

### OpenCL C Language

- `size_t` is the unsigned version of `int`.
- `T` can be type long, ulong, or double.
- `T` is type int, uint, or uint if supported.
- `T` can also be type long or `long long`.
- `T` is type `uchar`, `short`, or `int`.
- `T` can also be type `uchar`, `short`, or `int`.
- `T` can also be type `uchar`, `short`, or `int`.
- `T` can also be type `uchar`, `short`, or `int`.

Example:

```c
T o = 0x12345678;
T a = 0x12345678;
T b = 0x98765432;
T c = 0x98765432;
T d = 0x98765432;
T e = 0x98765432;
T f = 0x98765432;
T g = 0x98765432;
T h = 0x98765432;
T i = 0x98765432;
T j = 0x98765432;
T k = 0x98765432;
T l = 0x98765432;
T m = 0x98765432;
T n = 0x98765432;
T o = 0x98765432;
T p = 0x98765432;
T q = 0x98765432;
T r = 0x98765432;
T s = 0x98765432;
T t = 0x98765432;
T u = 0x98765432;
T v = 0x98765432;
T w = 0x98765432;
T x = 0x98765432;
T y = 0x98765432;
T z = 0x98765432;
```
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.

```c
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]);
```

Work-items in a work-group must execute this before any can continue.

Work-items in a sub-group must execute this before any can continue. Requires
the __opencl_c_subgroups feature.

Miscellaneous Vector Functions [C 6.13.12]

Tn and Tm are type char, uchar, short, shortn, ushort, ushortn, int, intr, uint, uintn, float, or
floatn. If supported, T can also be type long, longn, ulong, ulongn, double, or
doublel where n is 2, 4, 8, or 16 except in vector capped it may also be 3. Tn is uchar, USHORT, uint, uintn,
or ulongn. In all types listed here, long, ulong, or ulongn available only if supported.

```c
int vec_step(Tn a);
int vec_step (typename);
```

This built-in scalar or vector data type argument. Returns 1 for
scalar, 4 for 4-component vector, else number of elements in
the specified type.

```c
Tv shuffle (Ty x, Ty m);
Tv shuffle2 (Tv x, Ty y, Ty m);
```

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.

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, atomic_long, and atomic_ulong types are available if supported. The default
scope is memory_scope work_group for local atomics and memory_space_device for global atomics.

The default scope is memory_scope work_group for local atomics and memory_space_device
for global atomics, therefore the non-explicit functions require OpenCL C 2.0 or both the features
__opencl_c_atomic_scope_device and __opencl_c_atomic_scope_all_svm_devices.

The atomic object pointer supports the global and local address spaces. The expected pointer
supports the global, local, and private address spaces. For both pointers, the generic address
space is supported with the __opencl_c_generic_address_space feature.

```c
void atomic_init(volatile A *obj, C value);
void atomic_work_item_fence(cl_mem_fence_flags flags, memory_order order, memory_space scope);
void atomic_store(volatile A *obj, C desired, memory_order order [, memory_scope scope]);
void atomic_load(volatile A *obj, memory_order order [, memory_scope scope]);
C atomic_load_explicit(volatile A *obj, memory_order order [, memory_scope scope]);
C atomic_exchange(volatile A *obj, C desired, memory_order order [, memory_scope scope]);
C atomic_exchange_explicit(volatile A *obj, C desired, memory_order order [, memory_scope scope]);
bool atomic_compare_exchange_strong(volatile A *obj, C expected, C desired, memory_order order, memory_scope scope);
bool atomic_compare_exchange_strong_explicit(volatile A *obj, C expected, C desired, memory_order order, memory_scope scope);
bool atomic_compare_exchange_weak(volatile A *obj, C expected, C desired, memory_order order, memory_scope scope);
bool atomic_compare_exchange_weak_explicit(volatile A *obj, C expected, C desired, memory_order order, memory_scope scope);
```

Initializes the atomic object pointed to by obj to the value.

Effects based on value of order. flags must be
CLK_GLOBAL, LOCAL, IMAGE, MEM_FENCE or a combination of these.

Atomically replace the value pointed to by obj with the value of desired. Memory is affected
according to the value of order.

Atomically returns the value pointed to by obj. Memory is affected according to the value of order.

Atomically replace the value pointed to by obj with desired. Memory is affected
according to the value of order.

Atomically compares the value pointed to by obj for equality with that in expected, and if
true, replaces the value pointed to by obj with desired, and if false, updates the value in
expected with the value pointed to by obj. These operations are atomic read-modify-
write operations.

Atomically replaces the value pointed to by obj with the result of the computation
applied to the value pointed to by obj and the given operand.

Async Copies and Prefetch [C 6.13.10]

T is type char, char, uchar, uchar, short, shortn, ushort, ushortn, int, intr, uint, uintn, float, or
floatn. If supported, T can also be type long, longn, ulong, ulongn, double, or
doublel where n is 2, 4, 8, or 16 except in vector capped it may also be 3. Tn is uchar, USHORT, uint, uintn,
or ulongn. In all types listed here, long, ulong, or ulongn available only if supported.

```c
event_t async_work_group_copy ( __local T *dst,
const_global T *src, size_t num_gentypes, event_t event);
```

Copies num_gentypes T elements from src to dst.

```c
event_t async_work_group_strided_copy ( __local T *dst,
const_global T *src, size_t src_stride, event_t event);
```

```c
event_t async_work_group_strided_copy ( __local T *dst,
const __global T *src, size_t num_gentypes, event_t event);
```

Wait for completion of async_work_group_copy

```c
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.

```c
[const] global T * to_global ([const] T *p);
[const] local T * to_local ([const] T *p);
[const] private T * to_private ([const] T *p);
[const] cl_mem_fence_flags get_fence ( [const] T *p);
```

Memory fence value:

CLK_GLOBAL_MEM_FENCE, CLK_IMAGE_MEM_FENCE, CLK_LOCAL_MEM_FENCE.

```c
bool atomic_flag_test_and_set (volatile atomic_flag *obj);
bool atomic_flag_test_and_set_explicit(volatile atomic_flag *obj, memory_order order, memory_scope scope);
void atomic_flag_clear (volatile atomic_flag *obj);
void atomic_flag_clear_explicit(volatile atomic_flag *obj, memory_order order, memory_scope scope);
```

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.

```
atomic_flag_test_and_set (volatile atomic_flag *obj, memory_order order, memory_scope scope);
```

Atomically sets the value pointed to by obj 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

<table>
<thead>
<tr>
<th>key</th>
<th>op</th>
<th>computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>sub</td>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td>bitwise inclusive or</td>
</tr>
<tr>
<td>xor</td>
<td>^</td>
<td>bitwise exclusive or</td>
</tr>
</tbody>
</table>

Atomic Types and Enum Constants

Parameter type: memory_order

Values | Optional requirements |
---|---|
memory_order_relaxed | With any built-in atomic function except
memory_order_acq_rel | Requires OpenCL C 2.0 or support for
memory_order_seq_cst | Requires OpenCL C 2.0 or support for

Parameter type: memory_space

Values | Optional requirements |
---|---|
memory_space_work_group | Only used with atomic_work_item_fence with flags:
memory_space_work_item | Requires support for the __opencl_c_subgroups feature
memory_space_sub_group | Requires support for the __opencl_c_atomic_scope_all_svm_devices feature
memory_space_device | Requires support for the __opencl_c_atomic_scope_all_svm_devices feature
memory_space_all_svm_devices | Requires support for the __opencl_c_atomic_scope_all_svm_devices feature

(Continued on next page)
Atomic Functions (continued)

Atomic macros

- `atomic_int` (C value)
- `atomic_uint` (C value)
- `atomic_flag` (C value)
- `atomic_float` (C value)

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

```c
__opencl_c_pipes
__opencl_c_subgroups
```

Atomic integer and floating-point types

- `atomic_int` is supported by a limited subset of atomic operations.
- `atomic_int` size depends on whether implemented on 64-bit or 32-bit architecture.
- `atomic_flag` is supported only with these extensions enabled: `cl_khr_int64_base_atomics` and `cl_khr_int64_extended_atomics`.

Legacy Atomic Functions

These 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 workflow and, if supported, type long or ulong for extended features.

```c
atomic_flag
atomic_flag_init
atomic_flag_clear
atomic_flag_test_and_set
atomic_flag_compare_exchange
atomic_flag_compare_exchange_weak
atomic_flag_compare_exchange_strong
atomic_flag_signal
atomic_flag_wait
atomic_flag_signal_and_wait
atomic_flag_signal_and_wait_with_fences
atomic_flag_signal_or_wait
atomic_flag_wait_with_fences
atomic_flag_signal_or_wait_with_fences
```

Pipe Built-in Functions

`T` represents the built-in OpenCL C scalar or vector integer or floating-point 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 `CL_PLATFORM_EXTENSIONS` includes the OpenCL C 2.0 and 3.0 features.

```c
sub_group_scan_<op>_reduce
sub_group_scan_<op>_inclusive_reduce
sub_group_scan_<op>_exclusive_reduce
sub_group_scan_<op>_broadcast
```

Work-group Functions

- `atomic_xchg`
- `atomic_inc`
- `atomic_dec`
- `atomic_add`
- `atomic_sub`
- `atomic_cmpxchg`
- `atomic_min`
- `atomic_max`
- `atomic_and`
- `atomic_or`
- `atomic_xor`

Examples:

```c
int f = 1.00,2.00,3.00,4.00;
void *p = malloc(sizeof(int) * 4);
```

Print Function

```c
printf
```

Notes

- `atomic_xchg`
- `atomic_inc`
- `atomic_dec`
- `atomic_add`
- `atomic_sub`
- `atomic_cmpxchg`
- `atomic_min`
- `atomic_max`
- `atomic_and`
- `atomic_or`
- `atomic_xor`

Examples:

```c
float f = 1.00,2.00,3.00,4.00;
```

References:

- OpenCL 3.0 Reference Guide
- www.khronos.org/opencl

©2022 Khronos Group - Rev. 0922
The unnamed generic address space.

Functions require support for the features __opencl_c_sub_groups and __opencl_c_device_enqueue. All other functions require support for __opencl_c_device_enqueue or OpenCL C 2.0.

void retain_event(clk_event_t event)

void release_event(clk_event_t event)

clk_event_t create_event()

bool is_valid_event(clk_event_t event)

uint capture_event_profiling_info(
    clk_event_t event,
    clk_event_t *event_wait_list,
    const ndrange_t *ndrange,
    uint num_events_in_wait_list,
    int num_events_in_wait_list,
    const clk_event_t *event_wait_list,
    clk_event_t *event_wait_list,
    const ndrange_t *ndrange,
    void (*callback)(local void*, ..., ...),
    uint size, ...
)

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_QUEUE_FLAGS_NO_WAIT, WAIT_KERNEL, WAIT_WORK_GROUP.

Helper Built-in Functions

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

uint get_kernel_work_group_size(int block)

uint get_kernel_work_group_size(void *block[local void *])

uint get_kernel_preferred_work_group_size_multiple(clk_event_t *event_wait_list, const clk_event_t *event_wait_list, void (*callback)(local void*), ..., ...)

uint get_kernel_sub_group_count_for_ndrange(const ndrange_t ndrange, void (*callback)(void))

uint get_kernel_max_sub_group_size_for_ndrange(const ndrange_t ndrange, void (*callback)(void))

void is_valid_event(clk_event_t event)

void set_user_event_status(clk_event_t event, int status)

void enqueue_marker(queue_t queue, int num_events_in_wait_list, const ndrange_t *ndrange, const ndrange_t *ndrange, uint num_events_in_wait_list, const clk_event_t *event_wait_list, void (*callback)(local void*), ..., ...)

Event Built-in Functions

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

host

ocl_queue_t get_queue()

ocl_queue_t get_default_queue()

queue_t get_default_queue(void)

Default queue or CLK_NULL_QUEUE

builds a 1D ND-range descriptor.

builds a 2D or 3D ND-range descriptor. n may be 2 or 3.

Features and Feature Macros

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

Feature | The OpenCL 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_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_fp64 | Types and built-in functions with 64-bit floating point types.
__opencl_c_generic_address_space | The unnamed generic address space.
__opencl_c_images | Types and built-in functions for images.
__opencl_c_int64 | Types and built-in functions with 64-bit integers.
__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_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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Global</th>
<th>Constant</th>
<th>Local</th>
<th>Private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Dynamic allocation, Read/Write access</td>
<td>Dynamic allocation, Read/Write access</td>
<td>Dynamic allocation, No access</td>
<td>No allocation, No access</td>
</tr>
<tr>
<td>Kernel</td>
<td>No allocation, Read/Write access</td>
<td>Static allocation, Read-only access</td>
<td>Static allocation, Read/Write access</td>
<td>Static allocation, Read/Write access</td>
</tr>
</tbody>
</table>

The conceptual OpenCL device architecture 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.