OpenCL Programming Guide

Aaftab Munshi • Benedict R. Gaster
Timothy G. Mattson • James Fung • Dan Ginsburg

Foreword by Professor Pat Hanrahan, Stanford University

OpenCL Programming Guide

Work-Item Functions

Applications queue data-parallel and task-parallel kernels in OpenCL using the `clEnqueueNDRangeKernel` and `clEnqueueTask` APIs. For a data-parallel kernel that is queued for execution using `clEnqueue-NDRangeKernel`, an application specifies the global work size—the total number of work-items that can execute this kernel in parallel—and local work size—the number of work-items to be grouped together in a work-group. Table 5.1 describes the built-in functions that can be called by an OpenCL kernel to obtain information about work-items and work-groups such as the work-item's global and local ID or the global and local work size.

Figure 5.1 gives an example of how the global and local work sizes specified in `clEnqueueNDRangeKernel` can be accessed by a kernel executing on the device. In this example, a kernel is executed over a global work size of 16 items and a work-group size of 8 items per group.

OpenCL does not describe how the global and local IDs map to work-items and work-groups. An application, for example, cannot assume that a work-group whose group ID is `0` will contain work-items with global IDs `0 ... get_local_size(0) - 1`. This mapping is determined by the OpenCL implementation and the device on which the kernel is executing.

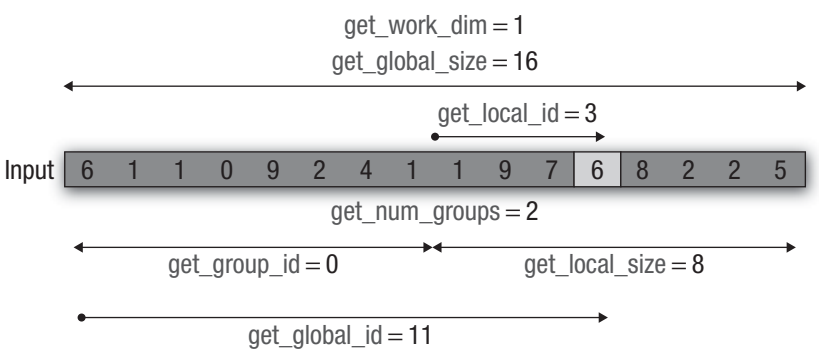

Figure 5.1 Example of the work-item functions

Table 5.1 Built-In Work-Item Functions

Function	Description
`uint` **`get_work_dim`**`()`	Returns the number of dimensions in use. This is the value given to the `work_dim` argument specified in `clEnqueueNDRangeKernel`. For `clEnqueueTask`, this function returns `1`.
`size_t` **`get_global_size`**`(uint` *`dimindx`*`)`	Returns the number of global work-items specified for the dimension identified by *`dimindx`*. This value is given by the `global_work_size` argument to `clEnqueueNDRangeKernel`. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_global_size()` returns `1`. For `clEnqueueTask`, this function always returns `1`.
`size_t` **`get_global_id`**`(uint` *`dimindx`*`)`	Returns the unique global work-item ID value for the dimension identified by *`dimindx`*. The global work-item ID specifies the work-item ID based on the number of global work-items specified to execute the kernel. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_global_id()` returns `0`. For `clEnqueueTask`, this function always returns `0`.
`size_t` **`get_local_size`**`(uint` *`dimindx`*`)`	Returns the number of local work-items specified for the dimension identified by *`dimindx`*. This value is given by the `local_work_size` argument to `clEnqueueNDRangeKernel` if `local_work_size` is not `NULL`; otherwise the OpenCL implementation chooses an appropriate `local_work_size` value. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_local_size()` returns `1`. For `clEnqueueTask`, this function always returns `1`.

continues

Table 5.1 Built-In Work-Item Functions (*Continued*)

Function	Description
`size_t` **`get_local_id`**`(uint` *`dimindx`*`)`	Returns the unique local work-item ID value, i.e., a work-item within a specific work-group for the dimension identified by *`dimindx`*. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_local_id()` returns `0`. For `clEnqueueTask`, this function always returns `0`.
`size_t` **`get_num_groups`**`(uint` *`dimindx`*`)`	Returns the number of work-groups that will execute a kernel for the dimension identified by *`dimindx`*. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_num_groups()` returns `1`. For `clEnqueueTask`, this function always returns `1`.
`size_t` **`get_group_id`**`(uint` *`dimindx`*`)`	Returns the work-group ID, which is a number from `0` to `get_num_groups(`*`dimindx`*`) - 1`. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_group_id()` returns `0`. For `clEnqueueTask`, this function always returns `0`.
`size_t` **`get_global_offset`**`(uint` *`dimindx`*`)`	Returns the offset values specified in the `global_work_offset` argument to `clEnqueueNDRangeKernel`. Valid values of *`dimindx`* are `0` to `get_work_dim() - 1`. For other values of *`dimindx`*, `get_global_offset()` returns `0`. For `clEnqueueTask`, this function always returns `0`.

Math Functions

OpenCL C implements the math functions described in the C99 specification. Applications that want to use these math functions include the `math.h` header in their codes. These math functions are available as built-ins to OpenCL kernels.[1]

We use the generic type name `gentype` to indicate that the math functions in Tables 5.2 and 5.3 take `float`, `float2`, `float3`, `float4`, `float8`, `float16`, and, if the double-precision extension is supported, `double`, `double2`, `double3`, `double4`, `double8`, or `double16` as the type for the arguments. The generic type name `gentypei` refers to the `int`, `int2`, `int3`, `int4`, `int8`, or `int16` data types. The generic type name `gentypef` refers to the `float`, `float2`, `float3`, `float4`, `float8`, or `float16` data types. The generic type name `gentyped` refers to the `double`, `double2`, `double3`, `double4`, `double8`, or `double16` data types.

In addition to the math functions listed in Table 5.2, OpenCL C also implements two additional variants of the most commonly used math functions for single-precision floating-point scalar and vector data types. These additional math functions (described in Table 5.3) trade accuracy for performance and provide developers with options to make appropriate choices. These math functions can be categorized as

- A subset of functions from Table 5.2 defined with the `half_` prefix. These functions are implemented with a minimum of 10 bits of accuracy, that is, a `ulp` value <= 8192 `ulp`.
- A subset of functions from Table 5.2 defined with the `native_` prefix. These functions typically have the best performance compared to the corresponding functions without the `native_` prefix or with the `half_` prefix. The accuracy (and in some cases the input ranges) of these functions is implementation-defined.
- `half_` and `native_` functions for the following basic operations: divide and reciprocal.

[1] The `math.h` header does not need to be included in the OpenCL kernel.

Table 5.2 Built-In Math Functions

Function	Description
gentype **acos**(gentype *x*)	Compute the arc cosine of *x*.
gentype **acosh**(gentype *x*)	Compute the inverse hyperbolic cosine of *x*.
gentype **acospi**(gentype *x*)	Compute acos(*x*)/p.
gentype **asin**(gentype *x*)	Compute the arc sine of *x*.
gentype **asinh**(gentype *x*)	Compute the inverse hyperbolic sine of *x*.
gentype **asinpi**(gentype *x*)	Compute asin(*x*)/p.
gentype **atan**(gentype *y_over_x*)	Compute the arc tangent of *y_over_x*.
gentype **atan2**(gentype *y*, gentype *x*)	Compute the arc tangent of *y*/*x*.
gentype **atanh**(gentype *x*)	Compute the hyperbolic arc tangent of *x*.
gentype **atanpi**(gentype *x*)	Compute atan(*x*)/p.
gentype **atan2pi**(gentype *y*, gentype *x*)	Compute atan2(*y*, *c*)/p.
gentype **cbrt**(gentype *x*)	Compute the cube root of *x*.
gentype ceil(gentype x)	Round to an integral value using the round-to-positive-infinity rounding mode.
gentype **copysign**(gentype *x*, gentype *y*)	Returns *x* with its sign changed to match the sign of *y*.
gentype **cos**(gentype *x*)	Compute the cosine of *x*.
gentype **cosh**(gentype *x*)	Compute the hyperbolic cosine of *x*.
gentype **cospi**(gentype *x*)	Compute cos(p*x*).

Function	Description
gentype **erfc**(gentype *x*)	Compute the complementary error function `1.0 - erf(x)`.
gentype **erf**(gentype *x*)	Compute the error function. For argument *x* this is defined as $\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$
gentype **exp**(gentype *x*)	Compute the base-e exponential of *x*.
gentype **exp2**(gentype *x*)	Compute the base-2 exponential of *x*.
gentype **exp10**(gentype *x*)	Compute the base-10 exponential of *x*.
gentype **expm1**(gentype *x*)	Compute $e^x - 1.0$.
gentype **fabs**(gentype *x*)	Compute the absolute value of a floating-point number.
gentype **fdim**(gentype *x*,gentype *y*)	Returns *x* - *y* if *x* > *y*, +0 if *x* is less than or equal to *y*.
gentype **floor**(gentype *x*)	Round to an integral value using the round-to-negative-infinity rounding mode.
gentype **fma**(gentype *a*, gentype *b*, gentype *c*)	Returns the correctly rounded floating-point representation of the sum of *c* with the infinitely precise product of *a* and *b*. Rounding of intermediate products does not occur. Edge case behavior is per the IEEE 754-2008 standard.
gentype **fmax**(gentype *x*,gentype *y*) gentypef **fmax**(gentypef *x*, float *y*) gentyped **fmax**(gentyped *x*, double *y*)	Returns *y* if *x* < *y*; otherwise it returns *x*. If one argument is a NaN, **fmax()** returns the other argument. If both arguments are NaNs, **fmax()** returns a NaN.

continues