OpenCL Events

An event is an object that communicates the status of OpenCL commands.

From the OpenCL Notes:
11. Enqueue the Kernel Object for Execution

- \( \text{size}[3] = \{ \text{NUM ELEMENT, 1, 1} \} \)
- \( \text{localWorkSize}[3] = \{ \text{LOCAL SIZE, 1, 1} \} \)
- \( \text{status} = \text{clEnqueueNDRangeKernel} \( \text{cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL} \) \)

Creating an Event

- \( \text{cl_event waitKernelC;} \)
- \( \text{status} = \text{clEnqueueNDRangeKernel} \( \text{cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, \&\text{waitKernelC}} \) \)

Waiting for Events

- \( \text{cl_event waitKernelA, waitKernelB}. \)
- \( \text{dependencies[2]} = \{ \text{waitKernelA, waitKernelB} \} \)
- \( \text{status} = \text{clEnqueueNDRangeKernel} \( \text{cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, NULL} \) \)

Waiting for One Event

- \( \text{cl_event waitKernelA, waitKernelB}. \)
- \( \text{dependencies[1]} = \{ \text{waitKernelA} \} \)
- \( \text{status} = \text{clEnqueueNDRangeKernel} \( \text{cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 1, dependencies, NULL} \) \)
Placing a Barrier in the Command Queue

```c
status = clEnqueueBarrier( cmdQueue );
```

This does not complete until all commands enqueued before it have completed.

Placing an Event Marker in the Command Queue

```c
cl_event waitMarker;
status = clEnqueueMarker( cmdQueue, &waitMarker );
```

This does not complete until all commands enqueued before it have completed.

This is just like a barrier, but it can throw an event to be waited for.

Waiting for Events Without Enqueuing Another Command

```c
status = clWaitForEvents( 2, dependencies );
```

```c
This blocks until the specified events are thrown, so use it carefully!
```

I Like Doing This

```c
// wait until all queued tasks have taken place:
void Wait( cl_command_queue queue )
{
    cl_event wait;
    cl_int status;
    status = clEnqueueMarker( queue, &wait );
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clEnqueueMarker failed
" );
    status = clWaitForEvents( 1, &wait );       // blocks until everything is done!
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clWaitForEvents failed"
" );
}
```

Call this before starting the timer, before ending the timer, and before using data from an array returned from OpenCL.

Getting Various Configuration IDs: Remember This?

```c
dl_uint numPlatforms;
status = clGetPlatformIDs( 0, NULL, &numPlatforms );
fprintf(stderr, "Number of Platforms = %d
", numPlatforms);
dl_platform_id * platforms = new dl_platform_id[numPlatforms ];
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

This way of querying information is a recurring OpenCL pattern.

Getting Event Statuses Works the Same Way

```c
cl_int eventStatus;
status = clGetEventInfo( waitKernelC, CL_EVENT_COMMAND_TYPE, sizeof(cl_int), &eventStatus, NULL );
```

Note that this is a nice way to check on event statuses without blocking. Thus, you could put this in a loop and go get some other work done in between calls.