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

```c
    size_t globalWorkSize[3] = { NUM_ELEMENT, 1, 1 };
    size_t localWorkSize[3] = { LOCAL_SIZE, 1, 1 };
    status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL );
```

Creating an Event

```c
    _d_event waitKernelC;
    status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, NULL );
```

Waiting for Events

```c
    _d_event waitKernelR, waitKernelB;
    ...
    if events event object
        event wait list
```

Creating an Execution Graph Structure

```c
    if events event object
        event(s) to wait for
```

```c
    status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, NULL );
```

```c
    if events event object
        event(s) to wait for
```

```c
    if events event object
        event(s) to wait for
```

```c
    status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, &waitKernelC );
```
cl_event waitKernelA, waitKernelB;
... 
status = clEnqueueNDRangeKernel(cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 1, &waitKernelA, NULL);
... 
waiting for one event

Placing a Barrier in the Command Queue

status = clEnqueueBarrier(cmdQueue);
Note: cannot throw an event
This does not complete until all commands enqueued before it have completed.

Placing an Event Marker in the Command Queue

cl_event waitMarker;
status = clEnqueueMarker(cmdQueue, &waitMarker);
Note: can throw an event
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

status = clWaitForEvents(2, dependencies);
This blocks until the specified events are thrown, so use it carefully!

I Like Doing Things This Way

// 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);
    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?

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

This way of querying information is a recurring OpenCL pattern

<table>
<thead>
<tr>
<th>How many to get</th>
<th>Where to put them</th>
<th>How many total there are</th>
</tr>
</thead>
<tbody>
<tr>
<td>status = clGetPlatformIDs(0, NULL, &amp;numPlatforms);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>status = clGetPlatformIDs(numPlatforms, platforms, NULL);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Getting Event Statuses Works the Same Way

<table>
<thead>
<tr>
<th>CL_EVENT_COMMAND_QUEUE</th>
<th>CL_EVENT_CONTEXT</th>
<th>CL_EVENT_COMMAND_TYPE</th>
<th>CL_EVENT_COMMAND_EXECUTION_STATUS</th>
</tr>
</thead>
</table>

Specify one of these:

- CL_QUEUE
- CL_SUBMITTED
- CL_RUNNING
- CL_COMPLETE

CL_EVENT_COMMAND_EXECUTION_STATUS returns one of these:

- CL_QUEUE
- CL_SUBMITTED
- CL_RUNNING
- CL_COMPLETE

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.

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