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
cl_event waitKernelC;
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelC );
```

# events event object

status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL );

cl_event waitKernelC;
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelC );
cl_event waitKernelA, waitKernelB.

... cl_event dependencies[2]; dependencies[0] = waitKernelA; dependencies[1] = waitKernelB;

status = clEnqueueNDRangeKernel( cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 1, &waitKernelA, NULL );

Waiting for Events

Creating an Execution Graph Structure

dr event waitKernelC;

... cl_event dependencies[2]; dependencies[0] = waitKernelA; dependencies[1] = waitKernelB;

status = clEnqueueNDRangeKernel( cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependencies, &waitKernelC );

Waiting for Events

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

```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\n" );

    status = clWaitForEvents( 1, &wait );  // blocks until everything is done!
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clWaitForEvents failed\n" );
}
```

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
cl_uint numPlatforms;
status = clGetPlatformIDs( 0, NULL, &numPlatforms );
free( stderr, "Number of Platforms = %d\n", 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>status = clGetPlatformIDs( numPlatforms, platforms, NULL );</td>
<td></td>
</tr>
</tbody>
</table>
Getting Event Statuses Works the Same Way

CL_EVENT_COMMAND_QUEUE
CL_EVENT_CONTEXT
CL_EVENT_COMMAND_TYPE
CL_EVENT_COMMAND_EXECUTION_STATUS

Specify one of these:

- CL_QUEUE
- CL_SUBMITTED
- CL_RUNNING
- CL_COMPLETE

Note that this 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.