OpenCL Events

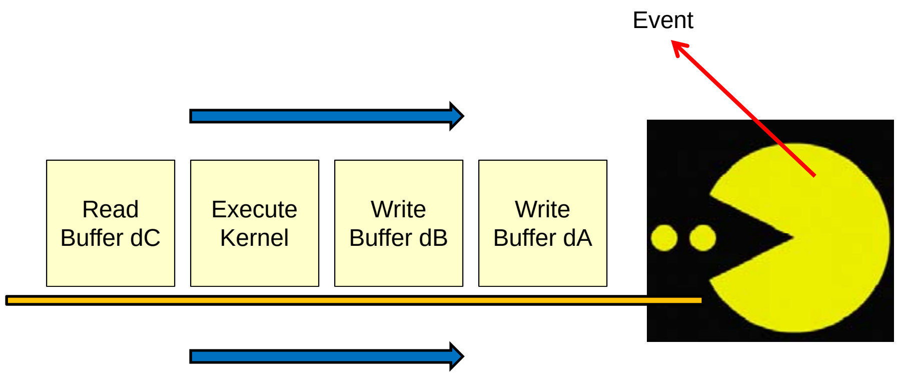
Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University

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

```
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 );
```

# events **event object**

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

event wait list

Creating an Event

event being created

```
cl_event   waitKernelC;

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

event(s) to wait for

Waiting for Events

```
cl_event  waitKernelA,  waitKernel B.

              . . .

cl_event  dependencies[ 2 ];

dependencies[ 0 ] = waitKernelA;
dependencies[ 1 ] = waitKernelB;

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

event being created

event(s) to wait for

Oregon State University
Computer Graphics

Waiting for One Event

```
cl_event  waitKernelA,  waitKernel B.

              . . .

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

Oregon State University
Computer Graphics

Placing a Barrier in the Command Queue

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

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 );
```

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 = clEnqueueWaitForEvents( cmdQueue,  2,  dependencies );
```

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

I Like Doing This

```
// 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 = clEnqueueWaitForEvents( queue, 1, &wait );
        if( status != CL_SUCCESS )
                fprintf( stderr, "Wait: clEnqueueWaitForEvents failed\n" );
        else
                fprintf( stderr, "Wait: clEnqueueWaitForEvents returned\n" );
}
```

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

Getting Event Statuses: Remember This?

```
cl_uint  numPlatforms;
status = clGetPlatformIDs( 0, NULL, &numPlatforms );

fprintf( 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

```
                                How many     Where to     How many total
                                 to get      put them       there are
status = clGetPlatformIDs(        0,           NULL,      &numPlatforms );
                                  |             |              |
status = clGetPlatformIDs( numPlatforms,    platforms,        NULL      );
```

Getting Event Statuses

```
cl_int  eventStatus;

status = clGetEventInfo( waitKernelC,  CL_EVENT_COMMAND_EXECUTION_STATUS,  1*sizeof(cl_int),
              &eventStatus,  NULL  );
```

CL_QUEUED

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..