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);

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

Creating an Event

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

# events to wait for before this kernel is allowed to execute

event wait list

event that will be thrown when this kernel is finished executing

event(s) to wait for before this kernel is allowed to execute

event that will be thrown when this kernel is finished executing

event(s) to wait for before this kernel is allowed to execute
Waiting for Events from Previously-Executed Kernels

d_event waitKernelA, waitKernelB, waitKernelC;

... 

d_event dependenciesAB[2]:
dependenciesAB[0] = waitKernelA;
dependenciesAB[1] = waitKernelB;
status = clEnqueueNDRangeKernel(cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependenciesAB, NULL);

Creating an Execution Graph Structure

d_event waitKernelA, waitKernelB, waitKernelC;
d_event dependenciesAB[2]:
dependenciesAB[0] = waitKernelA;
dependenciesAB[1] = waitKernelB;
status = clEnqueueNDRangeKernel(cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependenciesAB, &waitKernelC);

Creating the Full Execution Graph Structure

d_event waitKernelA, waitKernelB, waitKernelC, waitKernelD;
d_event dependenciesAB[2]:
dependenciesAB[0] = waitKernelA;
dependenciesAB[1] = waitKernelB;
d_event dependenciesCD[2]:
dependenciesCD[0] = waitKernelC;
dependenciesCD[1] = waitKernelD;
status = clEnqueueNDRangeKernel(cmdQueue, kernelA, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelA);
status = clEnqueueNDRangeKernel(cmdQueue, kernelB, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelB);
status = clEnqueueNDRangeKernel(cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependenciesAB, &waitKernelC);
status = clEnqueueNDRangeKernel(cmdQueue, kernelD, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelD);
status = clEnqueueNDRangeKernel(cmdQueue, kernelE, 1, NULL, globalWorkSize, localWorkSize, 2, dependenciesCD, NULL);

Waiting for One Event

d_event waitKernelA, waitKernelB;

... 

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

event(s) to wait for before this kernel is finished executing

Creating an Execution Graph Structure

event(s) to wait for before this kernel is allowed to execute

Creating the Full Execution Graph Structure

event(s) to wait for before this kernel is finished executing

Waiting for One Event

event(s) to wait for
Placing a Barrier in the Command Queue

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

Note: this cannot throw its own event

Placing an Event Marker in the Command Queue

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.

Note: this can throw its own event

Waiting for Events Without Enqueuing Another Command

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

I Like Synchronizing Things This Way

Call this before starting the timer, before ending the timer, and before retrieving data from an array computed in an OpenCL program.
Getting Event Statuses Without Blocking

CL_EVENT_COMMAND_QUEUE
CL_EVENT_CONTEXT
CL_EVENT_COMMAND_TYPE
CL_EVENT_COMMAND_EXECUTION_STATUS

Specify one of these

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

CL_EVENT_COMMAND_EXECUTION_STATUS
returns one of these

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.