Parallel Computing with GPUs

CUDA Streams
Part 3 - Synchronisation

The
University
Of
Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- ❑Synchronisation
  - ❑Identify and explain different levels of synchronisation
  - ❑Introduce and give examples of events
  - ❑Demonstrate the use of callbacks

Explicit Device Synchronisation

- What if we want to ensure an asynchronous kernel call has completed?
  - For timing kernel execution
  - Accessing data copied asynchronously without causing race conditions

- `cudaDeviceSynchronize()`
  - Will ensure that all asynchronous device operations are completed
  - Synchronise everything!
- `cudaStreamSyncronize(stream)`
  - Blocks host until all calls in stream are complete
- *CUDA Event synchronisation…*

Events

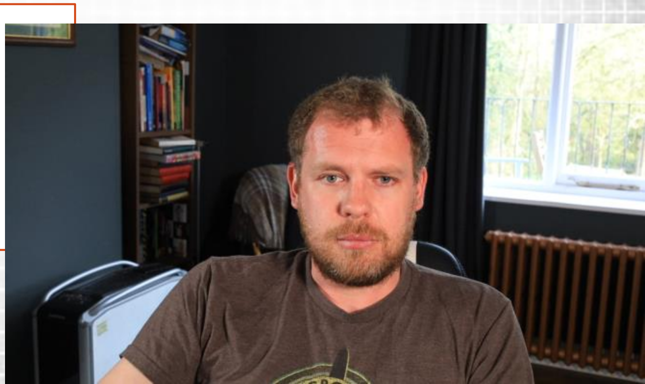
- Mechanism in which to signal when operations have occurred in a stream
    - Places an event into a stream (default stream unless specified)
- We have seen events already!
    - When timing our code…

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start);
my_kernel <<<(N /TPB), TPB >>>();
cudaEventRecord(stop);

cudaEventSynchronize(stop);
float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, stop);

cudaEventDestroy(start);
cudaEventDestroy(stop);
```

Events and Streams

- `cudaEventRecord(event, stream)`
  - Places an event in the non default stream
- `cudaEventSynchronize(event)`
  - Blocks until the stream completes all outstanding calls
  - Should be called after the event is inserted into the stream
- `cudaStreamWaitEvent(stream, event)`
  - Blocks the stream until the event occurs
  - Only blocks launches after event
  - Does not block the host
- `cudaEventQuery(event, stream)`
  - Has the event occurred in the stream

```
cudaMemcpyAsync(d_in, in, size, H2D, stream1);
cudaEventRecord(event, stream1); // record event

cudaStreamWaitEvent(stream2, event); // wait for event in stream1
kernel << <BLOCKS, TPB, 0, stream2 >> > (d_in, d_out);
```



Callbacks

- Callbacks are functions on the host which should be called when an event is reached
- cudaStreamAddCallback(stream, callback, user_data, 0)
  - Good for scheduling host code once event has completed
  - Allows GPU to initiate operations that only the CPU can perform
    - Disk or network IO
    - System calls, etc.

```
void CUDART_CB MyCallback(void *data){
    //some host code
}

MyKernel << <BLOCKS, TPB, 0, stream >> >(d_i);
cudaStreamAddCallback(stream, MyCallback, (void*)d_i, 0);
```

WDDM Command Queues

- GPUs driving a display in windows use the Windows Display Driver Model Command Queues.
  - All CUDA calls (sync/async) are buffered within a WDDM *Command Buffer*
  - The *Command Buffer* will usually be flushed by
    - Forcing it by calling `cudaEventQuery(0)`
    - Issuing a synchronous call. E.g. a stream/device sync or synchronous memcpy
    - Waiting until it gets full (unpredictable)
    - Magic???
- Implications
  - Only things in the same command buffer can be concurrent
  - Stuff *might* not get queued into copy/compute engines as you would expect
  - `cudaEventElapsedTime` *may* not be accurate for asynchronous host timing

https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9957-using-cuda-on-windows.pdf

Summary

- Synchronisation
  - Identify and explain different levels of synchronisation
  - Introduce and give examples of events
  - Demonstrate the use of callbacks

- Next Lecture: Multi GPU Programming