L17: Asynchronous Concurrent Execution, Open GL Rendering

Administrative

- Midterm
  - In class April 5, open notes
  - Review notes, readings and Lecture 15
- Project Feedback
  - Everyone should have feedback from me
  - Follow up later today on a few responses
- Design Review
  - Intermediate assessment of progress on project (next slide)
  - Due Monday, April 12, 5PM
  - Sign up on doodle.com poll http://doodle.com/24rm4guxtw2kchwe
- Final projects
  - Poster session, April 28 (dry run April 26)
  - Final report, May 5

Design Reviews

- Goal is to see a solid plan for each project and make sure projects are on track
  - Plan to evolve project so that results guaranteed
  - Show at least one thing is working
  - How work is being divided among team members
- Major suggestions from proposals
  - Project complexity – break it down into smaller chunks with evolutionary strategy
  - Add references – what has been done before? Known algorithm? GPU implementation?
  - In some cases, claim no communication but it seems needed to me

Design Reviews

- Oral, 10-minute Q&A session (April 14 in class, April 13/14 office hours, or by appointment)
  - Each team member presents one part
  - Team should identify "lead" to present plan
- Three major parts:
  - I. Overview
  - Define computation and high-level mapping to GPU
  - II. Project Plan
  - The pieces and who is doing what.
  - What is done so far? (Make sure something is working by the design review)
  - III. Related Work
  - Prior sequential or parallel algorithms/implementations
  - Prior GPU implementations (or similar computations)
- Submit slides and written document revising proposal that covers these and cleans up anything missing from proposal.

Final Project Presentation

- Dry run on April 26
  - Easels, tape and poster board provided
  - Tape a set of Powerpoint slides to a standard 2'x3' poster, or bring your own poster.
- Poster session during class on April 28
  - Invite your friends, profs who helped you, etc.
- Final Report on Projects due May 5
  - Submit code
  - And written document, roughly 10 pages, based on earlier submission.
  - In addition to original proposal, include
    - Project Plan and How Decomposed (from DR)
    - Description of CUDA implementation
    - Performance Measurement
    - Related Work (from DR)

Sources for Today's Lecture

- Presentation (possibly related to particle project in SDK) http://www.nvidia.com/content/cudazone/download/Advanced_CUDA_Training_NVISION08.pdf

  This presentation also talks about finite differencing and molecular dynamics.
- Asynchronous copies in CUDA Software Developer Kit called asyncAPI, bandwidthTest
- Chapter 3.2.6 in CUDA 3.0 Programmer's Guide
- Chapter 3.2.7.1 for Open GL interoperability

Overview of Concurrent Execution

- Review semantics of kernel launch
- Key performance consideration of using GPU as accelerator?
  - COPY COST!!!
- Some applications are data intensive, and even large device memories are too small to hold data
  - Appears to be a feature of some of your projects, and probably of the MRI application we studied
- Concurrent operation available on newer GPUs
  - Overlapping Data Transfer and Kernel Execution
  - Concurrent Kernel Execution
  - Concurrent Data Transfers

Review from L2: Semantics of Kernel Launch

- Kernel launch is asynchronous (> CC 1.1), synchronize at end
- Timing example (excerpt from simpleStreams in CUDA SDK):

```
cudaEvent_t start_event, stop_event;
cudaEventCreate(&start_event);
cudaEventCreate(&stop_event);
cudaEventRecord(start_event, 0);
 init_array<<<blocks, threads>>>(d_a, d_c, niterations);
 cudaEventRecord(stop_event, 0);
 cudaEventSynchronize(stop_event);
 cudaEventElapsedTime(&elapsed_time, start_event, stop_event);
```

- A bunch of runs in a row example (excerpt from transpose in CUDA SDK)

```
for (int i = 0; i < numIterations; ++i) {
      transpose<<< grid, threads >>>(d_odata, d_idata, size_x, size_y);
}
cudaThreadSynchronize();
```

Optimizing Host-Device Transfers

- Host-Device Data Transfers
  - Device to host memory bandwidth much lower than device to device bandwidth
  - 8 GB/s peak (PCI-e x16 Gen 2) vs. 102 GB/s peak (Tesla C1060)
- Minimize transfers
  - Intermediate data can be allocated, operated on, and deallocated without ever copying them to host memory
- Group transfers
  - One large transfer much better than many small ones

Slide source: Nvidia, 2008

Asynchronous Copy To/From Host (compute capability 1.1 and above)

- Warning: I have not tried this, and could not find a lot of information on it.
- Concept:
  - Memory bandwidth can be a limiting factor on GPUs
  - Sometimes computation cost dominated by copy cost
  - But for some computations, data can be "tiled" and computation of tiles can proceed in parallel (some of our projects)
  - Can we be computing on one tile while copying another?
- Strategy:
  - Use page-locked memory on host, and asynchronous copies
  - Primitive **cudaMemcpyAsync**
  - Effect is GPU performs DMA from Host Memory
  - Synchronize with **cudaThreadSynchronize()**

Copying from Host to Device

- cudaMemcpy(dst, src, nBytes, direction)
  - Can only go as fast as the PCI-e bus and not eligible for asynchronous data transfer
- cudaMallocHost(…): Page-locked host memory
  - Use this in place of standard malloc(…) on the host
  - Prevents OS from paging host memory
  - Allows PCI-e DMA to run at full speed
- Asynchronous data transfer
  - Requires page-locked host memory
- Enables highest cudaMemcpy performance
  - 3.2 GB/s on PCI-e x16 Gen1
  - 5.2 GB/s on PCI-e x16 Gen2
- See bandwidthTest in SDK

What does Page-Locked Host Memory Mean?

- How the Async copy works:
  - DMA performed by GPU memory controller
  - CUDA driver takes virtual addresses and translates them to physical addresses
  - Then copies physical addresses onto GPU
  - Now what happens if the host OS decides to swap out the page???
- Special malloc holds page in place on host
  - Prevents host OS from moving the page
  - CudaMallocHost()
- But performance could degrade if this is done on lots of pages!
  - Bypassing virtual memory mechanisms

Example of Asynchronous Data Transfer

```
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
cudaMemcpyAsync(dst1, src1, size, dir, stream1);
kernel<<<grid, block, 0, stream1>>>(…);
cudaMemcpyAsync(dst2, src2, size, dir, stream2);
kernel<<<grid, block, 0, stream2>>>(…);
```

src1 and src2 must have been allocated using cudaMallocHost
stream1 and stream2 identify streams associated with asynchronous call (note 4th "parameter" to kernel invocation)

Code from asyncAPI SDK project

```
// allocate host memory
CUDA_SAFE_CALL( cudaMallocHost((void**)&a, nbytes) );
memset(a, 0, nbytes);

// allocate device memory
CUDA_SAFE_CALL( cudaMalloc((void**)&d_a, nbytes) );
CUDA_SAFE_CALL( cudaMemset(d_a, 255, nbytes) );

… // declare grid and thread dimensions and create start and stop events

// asynchronously issue work to the GPU (all to stream 0)
cudaEventRecord(start, 0);
cudaMemcpyAsync(d_a, a, nbytes, cudaMemcpyHostToDevice, 0);
increment_kernel<<<blocks, threads, 0, 0>>>(d_a, value);
cudaMemcpyAsync(a, d_a, nbytes, cudaMemcpyDeviceToHost, 0);
cudaEventRecord(stop, 0);

// have CPU do some work while waiting for GPU to finish

// release resources
CUDA_SAFE_CALL( cudaFreeHost(a) );
CUDA_SAFE_CALL( cudaFree(d_a) );
```


Tiling Idea

Use tiling to manage data too large to fit into GPU

```
for each pair of tiles (make sure not to go off end)
    if (not first iteration) synchronize for stream1
    Initiate copy for stream1 into GPU
    Launch kernel for stream1
    if (not first iteration) synchronize for stream2
    Initiate copy for stream2
    Launch kernel for stream2

 // Clean up
Last tile, final synchronization
```


Concurrent Kernel Execution (compute capability 2.0)

- NOTE: Not available in current systems!
- Execute multiple kernels concurrently
- Upper limit is 4 kernels at a time
- Keep in mind, sharing of all resources between kernels may limit concurrency and its profitability

Concurrent Data Transfers (compute capability 2.0)

- NOTE: Not available in current systems!
- Can concurrently copy from host to GPU and GPU to host using asynchronous Memcpy

A Few More Details

- Only available in "some" more recent architectures
- Does your device support cudaMemCpyAsync?
  - Call cudaGetDeviceProperties()
  - Check the deviceOverlap property
- Does your device support concurrent kernel execution
  - Call cudaGetDeviceProperties()
  - Check the concurrentKernels property

2. OpenGL Rendering

- OpenGL buffer objects can be mapped into the CUDA address space and then used as global memory
  - Vertex buffer objects
  - Pixel buffer objects
- Allows direct visualization of data from computation
  - No device to host transfer
  - Data stays in device memory –very fast compute / viz
  - Automatic DMA from Tesla to Quadro (via host for now)
- Data can be accessed from the kernel like any other global data (in device memory)

OpenGL Interoperability

1. Register a buffer object with CUDA
   - **cudaGLRegisterBufferObject(GLuintbuffObj);**
   - OpenGL can use a registered buffer only as a source
   - Unregister the buffer prior to rendering to it by OpenGL
2. Map the buffer object to CUDA memory
   - **cudaGLMapBufferObject(void**devPtr, GLuintbuffObj);**
   - Returns an address in global memory Buffer must be registered prior to mapping
3. Launch a CUDA kernel to process the buffer
- Unmap the buffer object prior to use by OpenGL
   - **cudaGLUnmapBufferObject(GLuintbuffObj);**
4. Unregister the buffer object
   - **cudaGLUnregisterBufferObject(GLuintbuffObj);**
   - Optional: needed if the buffer is a render target
5. Use the buffer object in OpenGL code

Example from simpleGL in SDK

1. GL calls to create and initialize buffer, then registered with CUDA:

```
// create buffer object
glGenBuffers( 1, vbo);
glBindBuffer( GL_ARRAY_BUFFER, *vbo);

// initialize buffer object
unsigned int size = mesh_width * mesh_height * 4 * sizeof( float)*2;
glBufferData( GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);
glBindBuffer( GL_ARRAY_BUFFER, 0);

// register buffer object with CUDA
cudaGLRegisterBufferObject(*vbo);
```


Example from simpleGL in SDK, cont.

2. Map OpenGL buffer object for writing from CUDA

```
float4 *dptr;
cudaGLMapBufferObject( (void**)&dptr, vbo));
```

3. Execute the kernel to compute values for dptr

```
dim3 block(8, 8, 1);
dim3 grid(mesh_width / block.x, mesh_height / block.y, 1);
kernel<<< grid, block>>>(dptr, mesh_width, mesh_height, anim);
```

4. Unregister the OpenGL buffer object and return to Open GL

```
cudaGLUnmapBufferObject( vbo);
```


Next Week

- Exam on Monday
- Sorting algorithms / Open CL?