CUDA Odds and Ends

Patrick Cozzi
University of Pennsylvania
CIS 565 - Spring 2011

Administrivia

- Assignment 5
  - Handed out Wednesday, 03/16
  - Due Friday, 03/25
- Project
  - One page pitch due Sunday, 03/20, at 11:59pm
  - 10 minute pitch in class on Monday, 03/21

Agenda

- Atomic Functions
- Paged-Locked Host Memory
- Streams
- Graphics Interoperability
Atomic Functions

What is the value of `count` if 8 threads execute `++count`?

```c
__device__ unsigned int count = 0;
// ...
++count;
```

Atomic Functions

Read-modify-write atomic operation
- Guaranteed no interference from other threads
- No guarantee on order
- Shared or global memory
- Requires compute capability 1.1 (> G80)

See G.1 in the NVIDIA CUDA C Programming Guide for full compute capability requirements

Atomic Functions

How do you implement `atomicInc` below?

```c
__device__ int atomicAdd(
    int *address, int val);
// atomic ++count
atomicInc(&count, 1);  
```
Atomic Functions

How do you implement \texttt{atomicInc}?

\begin{verbatim}
__device__ int atomicAdd(
    int *address, int val)
{ // Made up keyword:
    __lock (address) {
        *address += value;
    }
}
\end{verbatim}

Atomic Functions

How do you implement \texttt{atomicInc} \textbf{without} locking?

Atomic Functions

\texttt{atomicCAS} pseudo implementation

\begin{verbatim}
int atomicCAS(int *address, 
    int compare, int val)
{ // Made up keyword
    __lock(address) {
        int old = *address;
        *address = (old == compare) ? val : old;
        return old;
    }
}
\end{verbatim}
Atomic Functions

- **atomicCAS** pseudo implementation

```c
int atomicCAS(int *address,
    int compare, int val)
{
    // Made up keyword
    __lock(address) {
        int old = *address;
        *address = (old == compare) ? val : old;
        return old;
    }
}
```

Example:

*addr = 1;

```c
atomicCAS(addr, 1, 2);
atomicCAS(addr, 1, 3);
atomicCAS(addr, 2, 3);
```
Atomic Functions

Example:

```c
*addr = 1;
atomicCAS(addr, 1, 2);
atomicCAS(addr, 1, 3); // returns 2
atomicCAS(addr, 2, 3); // *addr = 2
```

Again, how do you implement `atomicInc` given `atomicCAS`?

```c
__device__ int atomicAdd(
    int *address, int val);
```

Atomic Functions

Example:

```c
*addr = 1;
atomicCAS(addr, 1, 2);
atomicCAS(addr, 1, 3);
atomicCAS(addr, 2, 3); // returns 2 // *addr = 3
```

__device__ int atomicAdd(int *address, int val) {
    int old = *address, assumed;
    do {
        assumed = old;
        old = atomicCAS(address, 
            assumed, val + assumed);
    } while (assumed != old);
    return old;
}`
Atomic Functions

__device__ int atomicAdd(int *address, int val)
{
    int old = *address, assumed;
    do {
        assumed = old;
        old = atomicCAS(address,
                        assumed, assumed + val);
    } while (assumed != old);
    return old;
}

Atomic Functions

__device__ int atomicAdd(int *address, int val)
{
    int old = *address, assumed;
    do {
        assumed = old;
        old = atomicCAS(address,
                        assumed, assumed + val);
    } while (assumed != old);
    return old;
}

Lots of atomics:

- Arithmetic
  - atomicAdd()
  - atomicSub()
  - atomicExch()
  - atomicXor()
  - atomicMin()
  - atomicMax()
  - atomicInc()
  - atomicDec()
  - atomicCAS()

- Bitwise
  - atomicAnd()
  - atomicOr()

See B.10 in the NVIDIA CUDA C Programming Guide.
Atomic Functions
- How can threads from different blocks work together?
- Use atomics sparingly. Why?

Page-Locked Host Memory
- **Page-locked Memory**
  - Host memory that is essentially removed from virtual memory
  - Also called *Pinned Memory*

**Page-Locked Host Memory**
- **Benefits**
  - Overlap kernel execution and data transfers

<table>
<thead>
<tr>
<th>Time</th>
<th>Data Transfer</th>
<th>Kernel Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normally:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paged-locked:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See G.1 in the NVIDIA CUDA C Programming Guide for full compute capability requirements.

Page-Locked Host Memory

**Benefits**
- **Writing-Combing Memory**
  - Page-locked memory is cacheable
  - Allocate with `cudaHostAllocWriteCombined`
    - Avoid polluting L1 and L2 caches
    - Avoid snooping transfers across PCIe
    - Improve transfer performance up to 40% - in theory
  - Reading from write-combing memory is slow!
    - Only write to it from the host

- Paged-locked host memory can be mapped into the address space of the **device** on some systems
  - What systems allow this?
  - What does this eliminate?
  - What applications does this enable?
  - Call `cudaGetDeviceProperties()` and check `canMapHostMemory`

**Usage:**
- `cudaHostAlloc()` / `cudaMallocHost()`
- `cudaHostFree()`
- `cudaMemcpyAsync()`

DEMO

CUDA SDK Example: bandwidthTest
Page-Locked Host Memory

- What’s the catch?
  - Page-locked memory is scarce
    - Allocations will start failing before allocation of in-pageable memory
  - Reduces amount of physical memory available to the OS for paging
    - Allocating too much will hurt overall system performance

Streams

- **Stream**: Sequence of commands that execute in order
- Streams may execute their commands out-of-order or concurrently with respect to other streams

**Streams**

<table>
<thead>
<tr>
<th>Stream A</th>
<th>Stream B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command 0</td>
<td>Command 0</td>
</tr>
<tr>
<td>Command 1</td>
<td>Command 1</td>
</tr>
<tr>
<td>Command 2</td>
<td>Command 2</td>
</tr>
</tbody>
</table>

**Is this a possible order?**

<table>
<thead>
<tr>
<th>Stream A</th>
<th>Stream B</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command 0</td>
<td>Command 0</td>
<td>Command 0</td>
</tr>
<tr>
<td>Command 1</td>
<td>Command 1</td>
<td>Command 1</td>
</tr>
<tr>
<td>Command 2</td>
<td>Command 2</td>
<td>Command 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command 0</th>
<th>Command 1</th>
<th>Command 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command 0</td>
<td>Command 1</td>
<td>Command 2</td>
</tr>
<tr>
<td>Command 2</td>
<td>Command 0</td>
<td>Command 1</td>
</tr>
</tbody>
</table>
Streams

Is this a possible order?

Stream A  Stream B
Command 0  Command 0
Command 1  Command 1
Command 2  Command 2

Streams

Is this a possible order?

Stream A  Stream B
Command 0  Command 0
Command 1  Command 1
Command 2  Command 2

Streams

Is this a possible order?

Stream A  Stream B
Command 0  Command 0
Command 1  Command 1
Command 2  Command 2

Streams

In CUDA, what commands go in a stream?
- Kernel launches
- Host→device memory transfers
Streams

- Code Example
  1. Create two streams
  2. Each stream:
     1. Copy page-locked memory to device
     2. Launch kernel
     3. Copy memory back to host
  3. Destroy streams

Stream Example (Step 1 of 3)

```c
cudaStream_t stream[2];
for (int i = 0; i < 2; ++i)
{
    cudaStreamCreate(&stream[i]);
}
float *hostPtr;
cudaMallocHost(&hostPtr, 2 * size);
```

Stream Example (Step 1 of 3)

```c
cudaStream_t stream[2];
for (int i = 0; i < 2; ++i)
{
    cudaStreamCreate(&stream[i]);
}
float *hostPtr;
cudaMallocHost(&hostPtr, 2 * size);
```

Stream Example (Step 1 of 3)

```c
cudaStream_t stream[2];
for (int i = 0; i < 2; ++i)
{
    cudaStreamCreate(&stream[i]);
}
float *hostPtr;
cudaMallocHost(&hostPtr, 2 * size);
```
for (int i = 0; i < 2; ++i)
{
    cudaMemcpyAsync(/* ... */,
            cudaMemcpyHostToDevice, stream[i]);
    kernel<<<100, 512, 0, stream[i]>>>
    (/* ... */);
    cudaMemcpyAsync(/* ... */,
            cudaMemcpyDeviceToHost, stream[i]);
    cudaMemcpyDeviceToHost, stream[i];
}

// Blocks until commands complete
cudaStreamDestroy(stream[i]);

Streams

- Assume compute capabilities:
  - Overlap of data transfer and kernel execution
  - Concurrent kernel execution
  - Concurrent data transfer
- How can the streams overlap?

See G.1 in the NVIDIA CUDA C Programming Guide for more on compute capabilities.
Streams

- Can we have more overlap than this?

Stream A
- Host to device memory
- Kernel execution
- Device to host memory

Stream B
- Host to device memory
- Kernel execution
- Device to host memory

Can we have this?

Streams

- Implicit Synchronization
  - An operation that requires a dependency check to see if a kernel finished executing:
    - Blocks all kernel launches from any stream until the checked kernel is finished

See 3.2.6.5.3 in the NVIDIA CUDA C Programming Guide for all limitations
Streams

- Performance Advice
  - Issue all independent commands before dependent ones
  - Delay synchronization (implicit or explicit) as long as possible

```c
// for (int i = 0; i < 2; ++i) // to device
cudaMemcpyAsync(/* ... */), stream[i]);

for (int i = 0; i < 2; ++i)
  kernel<<< /*... */ stream[i]>>()
;
for (int i = 0; i < 2; ++i) // to host
  cudaMemcpyAsync(/* ... */), stream[i]);
```

- Rewrite this to allow concurrent kernel execution

```c
for (int i = 0; i < 2; ++i)
{
  cudaMemcpyAsync(/* ... */), stream[i]);
  kernel<<< /*... */ stream[i]>>()
;
  cudaMemcpyAsync(/* ... */), stream[i]);
}
```

- Explicit Synchronization
  - cudaThreadSynchronize()
    - Blocks until commands in all streams finish
  - cudaStreamSynchronize()
    - Blocks until commands in a stream finish

See 3.2.6.5 in the NVIDIA CUDA C Programming Guide for more synchronization functions.
Timing with Stream Events

- *Events* can be added to a stream to monitor the device's progress.
- An event is completed when all commands in the stream preceding it complete.

```c
#include <cuda_runtime.h>

int main() {
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start, 0);
    for (int i = 0; i < 2; ++i) {
        // ...
        cudaEventRecord(stop, 0);
    }
    cudaEventSynchronize(stop);
    float elapsedTime;
    cudaEventElapsedTime(&elapsedTime, start, stop);
    // cudaEventDestroy(...)
}
```
Timing with Stream Events

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

cudaEventRecord(start, 0);
for (int i = 0; i < 2; ++i)
    // ...
cudaEventRecord(stop, 0);

float elapsedTime;
cudaEventSynchronize(stop);
cudaEventElapsedTime(&elapsedTime, start, stop);
// cudaEventDestroy(...
```

Timing with Stream Events

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

cudaEventRecord(start, 0);
for (int i = 0; i < 2; ++i)
    // ...
cudaEventRecord(stop, 0);

float elapsedTime;
cudaEventSynchronize(stop);
cudaEventElapsedTime(&elapsedTime, start, stop);
// cudaEventDestroy(...
```

Graphics Interoperability

- What applications use both CUDA and OpenGL/Direct3D?
  - CUDA ➔ GL
  - GL ➔ CUDA
- If CUDA and GL cannot share resources, what is the performance implication?

**Graphics Interop:** Map GL resource into CUDA address space
- Buffers
- Textures
- Renderbuffers
Graphics Interoperability

- OpenGL Buffer Interop
  1. Assign device with GL interop
     - cudaGLSetGLDevice()
  2. Register GL resource with CUDA
     - cudaGraphicsGLRegisterBuffer()
  3. Map it
     - cudaGraphicsMapResources()
  4. Get mapped pointer
     - cudaGraphicsResourceGetMappedPointer()