CUDA GPGPU Workshop 2012

Day-3:
Thread Cooperation in CUDA/C

Presenter(s):
Abu Asaduzzaman

Wichita State University
July 12, 2012
Important Notice

Tomorrow (Fri, 7/13): Starting at 9:00 AM

Tomorrow (Fri, 7/13): Starting at 9:00 AM

Tomorrow (Fri, 7/13): Starting at 9:00 AM
Important Notice

Materials of this Workshop are available on WSU Blackboard with the following title:
CUDA.org: CUDA GPGPU Workshop
CUDA GPGPU Workshop 2012

Outline
- Review: Day-1 and Day-2
- Thread Cooperation in CUDA/C
- Shared Memory and Synchronization
- Constant, Texture, and Page-Locked Host Memory
- Practice:
  - Vector Sums
  - Dot Product
  - Matrix Multiplication

QUESTIONS?
Any time, please.
Review: Day-1 and Day-2

(Workshop) Objectives
- To become a moderate to advanced level CUDA/C programmer
- To prepare pedagogy for future CSE courses
- To develop parallel computing research initiatives

Methodologies
- Discuss, study (book?), and practice
- CUDA Educator from Nvidia

(Workshop) Outcomes
- Understand the needs and benefits of parallel programming
- Write program in C, C thread, OpenMP/C, and Open MPI/C
- Understand NVIDIA GPU/CUDA technology
- Develop programs in CUDA/C for GPGPUs
## Workshop Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>9:30 am to 12:00 noon session</th>
<th>1:30 pm to 4:00 pm session</th>
</tr>
</thead>
<tbody>
<tr>
<td>July/10/2012</td>
<td>• Introduction to the Workshop&lt;br&gt;• Computing: past, present, and future&lt;br&gt;• GPGPU/CUDA/C and WSU&lt;br&gt;• Parallel Computing (by Nasrin)</td>
<td>• Practice&lt;br&gt;• C, C threads&lt;br&gt;• Open MP/MPI&lt;br&gt;• Open MPI (SMP, MPI)</td>
</tr>
<tr>
<td>Tuesday</td>
<td>(Asaduzzaman/WSU)</td>
<td></td>
</tr>
<tr>
<td>July/11/2012</td>
<td>• Brief history of GPGPU&lt;br&gt;• Intro to CUDA/C Programming&lt;br&gt;• CUDA Development Toolkit&lt;br&gt;• CUDA Arch &amp; Prog (by Chok)</td>
<td>• Practice&lt;br&gt;• Hello WSU!&lt;br&gt;• Summing vectors&lt;br&gt;• Fun example!</td>
</tr>
<tr>
<td>Wednesday</td>
<td>(Asaduzzaman/WSU)</td>
<td></td>
</tr>
<tr>
<td>July/12/2012</td>
<td>• <strong>Thread Cooperation in CUDA/C</strong>&lt;br&gt;• Shared memory and synchronization&lt;br&gt;• Texture, Page-Locked Host memory</td>
<td>• Practice&lt;br&gt;• Dot products&lt;br&gt;• Matrix multiplication</td>
</tr>
<tr>
<td>Thursday</td>
<td>(Asaduzzaman/WSU)</td>
<td></td>
</tr>
<tr>
<td>July/13/2012</td>
<td>• <strong>Advanced CUDA/C Programming</strong>&lt;br&gt;• CUDA Threads&lt;br&gt;• CUDA Memory&lt;br&gt;• Performance Considerations</td>
<td>• CUDA/C on multiple GPGPUs&lt;br&gt;• Virtualization on GPGPU&lt;br&gt;• Cloud Computing, MIMD/VLIW, and CUDA&lt;br&gt;• Thank you!</td>
</tr>
<tr>
<td>Friday</td>
<td>(Ebersole/NVIDIA)</td>
<td></td>
</tr>
</tbody>
</table>
Keywords

- `__global__ void kernel (void) {}`
  - Function kernel should be run on a device

- `addx<<<B, t/B>>>();`
  - Kernel call to `addx` generates B Blocks, each with t Threads
  - Total threads, \( T = B \times t \)

- `Grid`
  - Collection of parallel blocks
Keywords (cont’d)

- Unique Thread ID (tid)
  - tid = threadIdx.x + blockIdx.x * blockDim.x

- Warp
  - A group of threads that are being processed together
Review: Day-1 and Day-2 (5)

Let’s have fun!

- Compare
  - C thread and Open MP
  - Open MP and Open MPI
  - Open MPI and CUDA
  - OpenGL and CUDA
  - gcc and nvcc
  - Kernel and Grid
  - Kernel and Block
  - Block and Thread
  - Block, Thread, and DIM
Let’s have fun!

- NVIDIA, CUDA, and WSU
  - CUDA Teaching Center at WSU
Let’s have fun!

- Explain
  - CPU performance and GPU performance
  - GPU and GPGPU
  - NVIDIA and ATI Technology
  - Tesla C2075
    - 14 processors, 448 cores, 32 cores/proc
  - Quadro NVS 295
    - 1 processor, 8 cores
  - GeForce GTX 480
    - 6 processors, 480 cores, 80 cores/proc
Review: Day-1 and Day-2 (8)

Let’s have fun!

- Explain
  - Streaming Processors (SM)
    - 32 cores
    - 32K 32-bit registers
    - 64 KB Shared memory
    - 2 warp schedulers
    - Other function units
Let’s have fun!

- Explain
  - CUDA/C supports
    - Linux
    - Windows
  - North Bridge (NB), South Bridge (SB)
    - NB → faster devices (memory, GPU)
    - SB → slower devices
  - PCI and PCIe Buses
    - PCIe supports multiple GPUs
Review: Day-1 and Day-2 (10)

Let's have fun!

- Explain
  - Four steps (Host → Kernel → GPU)
    - Allocates memory (and/or copies data from Host to) in GPU
    - Host makes the kernel call
    - Copies data back to Host from GPU
    - Free allocated memory in GPU
Outline

- Review: Day-1 and Day-2
- Thread Cooperation in CUDA/C
- Shared Memory and Synchronization
- Constant, Texture, and Page-Locked Host Memory
- Practice:
  - Vector Sums
  - Dot Product
  - Matrix Multiplication

Q U E S T I O N S?
Any time, please.
Thread Cooperation

Kernel, Block, and Thread

- Host makes Kernel calls
  - Kernel → multiple Blocks
  - Block ID: 1D or 2D
  - Block → Threads
  - Thread ID: 1D, 2D, or 3D

`kernelx<<<B, T/B>>>( )`
`kernelx<<<4, 4>>>( ) ?`
`kernelx<<<2x3, 16x16>>>( ) ?`
`kernelx<<<1, 1025>>>( ) ?`
Thread Cooperation (2)

- Summing two vectors [1]
  - \( c[tid] = a[tid] + b[tid] \)
  - No dependency among threads

- Temperature of a bar [2]
  - Each point depends on 3 points from the previous step

[1] “CUDA by Example …” by Sanders, J. and Kandrot, E.
[2] “Effects of Number of MPI Parallel Processes on Simulation Run Time and Speedup” by Williams, C.
Thread Cooperation (3)

- In parallel computing
  - Processors share information
  - Parallel copies of code communicate
  - Processes cooperate to solve a problem

- CUDA/C mechanism
  - Splitting parallel Blocks
  - `addx<<<N, 1>>>( )`
  - N Blocks x 1 Threads/Block = N parallel Blocks
Thread Cooperation (4)

Vector Sums
(How CUDA/C splits parallel Blocks?)

- CPU Vector Sum

```c
void add( int *a, int *b, int *c ) {
    int tid = 0;   // this is CPU z
    while (tid < N) {
        c[tid] = a[tid] + b[tid];
        tid += 1;     // we have one C
    }
}
```

<table>
<thead>
<tr>
<th>CPU Core 1</th>
<th>CPU Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void add( int *a, int *b, int *c )</code></td>
<td><code>void add( int *a, int *b, int *c )</code></td>
</tr>
<tr>
<td><code>{</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td><code>    int tid = 0;</code></td>
<td><code>    int tid = 1;</code></td>
</tr>
<tr>
<td><code>    while (tid &lt; N) {</code></td>
<td><code>    while (tid &lt; N) {</code></td>
</tr>
<tr>
<td><code>        c[tid] = a[tid] +</code></td>
<td><code>        c[tid] = a[tid] +</code></td>
</tr>
<tr>
<td><code>        b[tid];</code></td>
<td><code>        b[tid];</code></td>
</tr>
<tr>
<td><code>    tid += 2;</code></td>
<td><code>    tid += 2;</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>
Thread Cooperation (5)

Vector Sums
(How CUDA/C splits parallel Blocks?)

- GPU Vector Sum (N Blocks)

```
add<<<N,1>>>( dev_a, dev_b, dev_c );

__global__ void add( int *a, int *b, int *c ) {
    int tid = blockIdx.x;  // handle the data index
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
```

- blockIdx.x – CUDA Blocks are 2-D (x-y)
Thread Cooperation (6)

Vector Sums
(How CUDA/C splits parallel Blocks?)

- GPU Vector Sum (N Threads)

```c
add<<<1,N>>>( dev_a, dev_b, dev_c );

__global__ void add( int *a, int *b, int *c ) {
    int tid = threadIdx.x;
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
```

- `threadIdx.x` – CUDA threads are 3-D (x-y-z)
Thread Cooperation (7)

Vector Sums
(How CUDA/C splits parallel Blocks?)

- Limits on Blocks and Threads
  - 65,353 Blocks; 512 Threads per Block

- 2-D Blocks and 3-D Threads

- Unique \( \text{tid} = \text{threadIdx.x} + \text{blockIdx.x} \times \text{blockDim.x} \)
  - \( \text{blockDim.x} \) (DIM) is the number of Threads per Block
  - \( \text{tid} = ? \)
  - \( \text{tid} = 10 \)

```c
add<<< (N+127)/128, 128 >>>( dev_a, dev_b, dev_c );
```
Thread Cooperation (8)

Vector Sums
(How CUDA/C splits parallel Blocks?)

- Long Vectors

```c
#define N (33 * 1024)

add<<<128,128>>>( dev_a, dev_b, dev_c );

__global__ void add( int *a, int *b, int *c ) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    while (tid < N) {
        c[tid] = a[tid] + b[tid];
        tid += blockDim.x * gridDim.x;
    }
}
```
Outline

- Review: Day-1 and Day-2
- Thread Cooperation in CUDA/C
- Shared Memory and Synchronization
- Constant, Texture, and Page-Locked Host Memory
- Practice:
  - Vector Sums
  - Dot Product
  - Matrix Multiplication

QUESTIONS?
Any time, please.
Shared Memory

- Shared memory (SM) resides on the GPU
  - Low latency
- Variables in SM are treated differently
  - One copy per Block
  - Threads cannot see or modify the copies within Blocks
  - Better communicate and collaborate within a Block
- Any problem with SM?
  - Race condition
- How to tackle race condition?
  - Synchronization
Synchronization

- Dot Product
  \[(x_1, x_2, x_3, \ldots) \cdot (y_1, y_2, y_3, \ldots) = x_1y_1 + x_2y_2 + x_3y_3 + \ldots\]

```c
const int N = 33 * 1024;
const int threadsPerBlock = 256;

__global__ void dot( float *a, float *b, float *c ) {
    __shared__ float cache[threadsPerBlock];
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int cacheIndex = threadIdx.x;
    float temp = 0;
    while (tid < N) {
        temp += a[tid] * b[tid];
        tid += blockDim.x * gridDim.x;
    }

    // set the cache values
    cache[cacheIndex] = temp;
    
    // Write back the cache values
    __syncthreads();
    cache[cacheIndex] = temp;
}
```
Synchronization (2)

- Dot Product
  - \((x_1, x_2, x_3, \ldots) \cdot (y_1, y_2, y_3, \ldots) = x_1y_1 + x_2y_2 + x_3y_3 + \ldots\)

```c
// synchronize threads in this block
__syncthreads();
```
Constant and Texture Memory

- GPU memory bandwidth bottleneck
  - Large number of arithmetic units
  - Large number of arithmetic throughputs

- Constant memory
  - GPU read-only memory (64KB)
  - Holds constant values
  - Faster access

- Texture memory
  - Spatial locality – a thread is likely to read from an address ‘near’ the address that nearby threads read
  - Non-consecutive threads can be cached together
  - Improved performance
Page-Locked Host Memory

- Data parallelism
  - SIMD

- Task parallelism
  - Two or more completely different tasks in parallel
  - Example: download from and upload to network

- CUDA stream
  - A queue of GPU tasks that gets executed in a specific order
  - Example: kernel launches and memory copies

- C malloc() Vs. cudaHostAlloc()
  - Page-able host memory Vs. page-locked host memory
Page-Locked Host Memory (2)

- Page-Locked Host memory
  - Aka, pinned memory
  - Always in physical/main memory; OS does not page it out
  - GPU can use direct memory access (DMA) to read/write data
Important Notice

Tomorrow (Fri, 7/13): Starting at 9:00 AM

Tomorrow (Fri, 7/13): Starting at 9:00 AM

Tomorrow (Fri, 7/13): Starting at 9:00 AM
Outline

- Review: Day-1 and Day-2
- Thread Cooperation in CUDA/C
- Shared Memory and Synchronization
- Constant, Texture, and Page-Locked Host Memory

Practice:
- Vector Sums
- Dot Product
- Matrix Multiplication

Questions?
Any time, please.
Practice

Vector Sums

- CPU
- GPU (Blocks)
- GPU (Threads)
  - user1@capplab22:~/CUDAWorkshop2012/Day_3$
Practice (2)

Dot Products

- CPU
- GPU (Blocks)
- GPU (Threads)
  - user1@capplab22:~/CUDAWorkshop2012/Day_3$
Matrix Multiplication

\[
\begin{pmatrix}
    a_{0,0} & a_{0,1} \\
    a_{1,0} & a_{1,1} \\
    a_{2,0} & a_{2,1} \\
    a_{3,0} & a_{3,1}
\end{pmatrix}
\times
\begin{pmatrix}
    b_{0,0} & b_{0,1} & b_{0,2} \\
    b_{1,0} & b_{1,1} & b_{1,2}
\end{pmatrix}
\]

\[
c_{0,0} = a_{0,0} b_{0,0} + a_{0,1} b_{1,0} \\
c_{0,1} = a_{0,0} b_{0,1} + a_{0,1} b_{1,1} \\
c_{0,2} = a_{0,0} b_{0,2} + a_{0,1} b_{1,2} \\
c_{1,0} = a_{1,0} b_{0,0} + a_{1,1} b_{1,0} \\
c_{1,1} = a_{1,0} b_{0,1} + a_{1,1} b_{1,1} \\
c_{1,2} = a_{1,0} b_{0,2} + a_{1,1} b_{1,2} \\
c_{2,0} = a_{2,0} b_{0,0} + a_{2,1} b_{1,0} \\
c_{2,1} = a_{2,0} b_{0,1} + a_{2,1} b_{1,1} \\
c_{2,2} = a_{2,0} b_{0,2} + a_{2,1} b_{1,2} \\
c_{3,0} = a_{3,0} b_{0,0} + a_{3,1} b_{1,0} \\
c_{3,1} = a_{3,0} b_{0,1} + a_{3,1} b_{1,1} \\
c_{3,2} = a_{3,0} b_{0,2} + a_{3,1} b_{1,2}
\]

drzaman@kirk:~/CUDAWorkshop2012/Day_1/Matrix$
Practice (4)

Matrix Multiplication

- C
- C thread
- Open MP/C
- Open MPI/C (SMP)
- Open MPI/C (MPI)
- CUDA/C
  - drzaman@kirk:~/CUDAWorkshop2012/Day_1/Matrix$
  - user1@capplab22:~/CUDAWorkshop2012/Day_3$
Practice (5)

More examples

- Single Stream
- Double Stream
- Time to copy
  - cudaHostAlloc( … )
  - cudaEventCreate( … )
  - user1@capplab22:~/CUDAWorkshop2012/Day_3$
Conclusions

Following topics are covered is Day-3:
- Thread cooperation in CUDA/C
- Shared memory and synchronization
- Texture and page-locked host memory
- Practice: Dot product, Matrix multiplication
- Performance analysis of various solutions

Topics for Day-4 (Final Day):
- Advanced CUDA/C Programming
- Advanced GPGPU/CUDA/C Research Topics
Questions?

☐ Any questions, comments, or suggestions?
GPGPU/CUDA/C Workshop 2012
Day-3: Thread Cooperation in CUDA/C

Thank you.

Please send your feedback to:
Abu Asaduzzaman (Dr. Zaman)
Tel: +1-316-978-5261
E-mail: Abu.Asaduzzaman@wichita.edu