HPCMP
PUPR-IPPCC 2011
LN1: CUDA Overview

S.V. Providence

Department of Computer Science
Hampton University
Hampton, Virginia 23668
stephen.providence@hamptonu.edu
(757)728-6406 (voice mail)

Polytechnic University of Puerto Rico
Intermediate Parallel Programming & Cluster Computing,
Thursday, Aug. 4\textsuperscript{th}, 2011
Outline

● Introduction
  ● GPU Hardware
  ● Programming Model
  ● Conclusion
Outline

- Introduction
- GPU Hardware
  - Programming Model
- Conclusion
Outline

- Introduction
- GPU Hardware
- Programming Model
- Conclusion
Outline

- Introduction
- GPU Hardware
- Programming Model
- Conclusion
Introduction
GPU

- 100s of cores
  - Programmable
  - Can be installed in most desktops

Figure: Tesla C1060

- Central to the second fastest computer on Earth (top500.org)
- Similar in price to CPU
Introduction

GPU

- 100s of cores
- Programmable
- Can be installed in most desktops

Figure: Tesla C1060

- Central to the second fastest computer on Earth (top500.org)
- Similar in price to CPU
Introduction

GPU

- 100s of cores
- Programmable
- Can be installed in most desktops

Figure: Tesla C1060

- Central to the second fastest computer on Earth (top500.org)
- Similar in price to CPU
Introduction

GPU

- 100s of cores
- Programmable
- Can be installed in most desktops

Figure: Tesla C1060

- Central to the second fastest computer on Earth (top500.org)
- Similar in price to CPU
Introduction

GPU

- 100s of cores
- Programmable
- Can be installed in most desktops

**Figure:** Tesla C1060

- Central to the second fastest computer on Earth (top500.org)
- Similar in price to CPU
Introduction

GPU

- 100s of cores
- Programmable
- Can be installed in most desktops

Figure: Tesla C1060

- Central to the second fastest computer on Earth (top500.org)
- Similar in price to CPU
Introduction

Performance

Figure: nvidia.com

GT200 = GeForce GTX 280
G71 = GeForce 7900 GTX
NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX
G70 = GeForce 7800 GTX
NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX
NV40 = GeForce 6800 Ultra
GPU H/W

μ-processor structure

Figure: nvidia.com
GPU H/W

µ-processor structure

Figure: nvidia.com
GPU H/W
µ-processor structure

- M procs w/ N cores ea. & dvgt threads may exe in parallel

SIMD - cores share IU w/ other cores in MP

Figure: nvidia.com
M procs w/ N cores ea. & dvgt threads may exe in parallel

Figure: nvidia.com

SIMD - cores share IU w/ other cores in MP
M procs w/ N cores ea. & dvgt threads may exe in parallel

SIMD - cores share IU w/ other cores in MP

Figure: nvidia.com
GPU H/W
mem. hier

- Procs have 32-bit regs & canst/text caches are R/O & are faster that shared mem

Figure: nvidia.com

- MPs have shared mem, const. & texture caches

Stephen V. Providence Ph.D.
High Performance Computing Modernization Program
Procs have 32-bit regs & canst/text caches are R/O & are faster that shared mem

Figure: nvidia.com

MPs have shared mem, const. & texture caches
Procs have 32-bit regs & canst/text caches are R/O & are faster than shared mem

Figure: nvidia.com

MPs have shared mem, const. & texture caches
933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
  - 3 multiprocessors per TPC
  - 8 cores per multiprocessor
  - 16384 registers per multiprocessor
  - 16 KB shared memory per multiprocessor
  - 64 KB constant cache per multiprocessor
  - 6 KB < texture cache < 8 KB per multiprocessor
  - 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

Figure: nvidia.com
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
  - 8 cores per multiprocessor
  - 16384 registers per multiprocessor
  - 16 KB shared memory per multiprocessor
  - 64 KB constant cache per multiprocessor
  - 6 KB < texture cache < 8 KB per multiprocessor
  - 1.3 GHz clock rate
  - Single and double-precision floating-point calculation
  - 1 GB DDR3 dedicated memory

Figure: nvidia.com
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
  - 16384 registers per multiprocessor
  - 16 KB shared memory per multiprocessor
  - 64 KB constant cache per multiprocessor
  - 6 KB < texture cache < 8 KB per multiprocessor
  - 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

Figure: nvidia.com
933 GFLOPS peak performance
10 thread processing clusters (TPC)
3 multiprocessors per TPC
8 cores per multiprocessor
16384 registers per multiprocessor
16 KB shared memory per multiprocessor
64 KB constant cache per multiprocessor
6 KB < texture cache < 8 KB per multiprocessor
1.3 GHz clock rate
Single and double-precision floating-point calculation
1 GB DDR3 dedicated memory

Figure: nvidia.com

Stephen V. Providence Ph.D.  High Performance Computing Modernization Program
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

Figure: nvidia.com
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

Figure: nvidia.com
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

Figure: nvidia.com
• 933 GFLOPS peak performance
• 10 thread processing clusters (TPC)
• 3 multiprocessors per TPC
• 8 cores per multiprocessor
• 16384 registers per multiprocessor
• 16 KB shared memory per multiprocessor
• 64 KB constant cache per multiprocessor
• 6 KB < texture cache < 8 KB per multiprocessor
• 1.3 GHz clock rate
• Single and double-precision floating-point calculation
• 1 GB DDR3 dedicated memory

Figure: nvidia.com

Stephen V. Providence Ph.D.  High Performance Computing Modernization Program
933 GFLOPS peak performance
10 thread processing clusters (TPC)
3 multiprocessors per TPC
8 cores per multiprocessor
16384 registers per multiprocessor
16 KB shared memory per multiprocessor
64 KB constant cache per multiprocessor
6 KB < texture cache < 8 KB per multiprocessor
1.3 GHz clock rate
Single and double-precision floating-point calculation
1 GB DDR3 dedicated memory
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

Figure: nvidia.com

Stephen V. Providence Ph.D.  High Performance Computing Modernization Program
GPU H/W
GTX 280 specs

- 933 GFLOPS peak performance
- 10 thread processing clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- 16384 registers per multiprocessor
- 16 KB shared memory per multiprocessor
- 64 KB constant cache per multiprocessor
- 6 KB < texture cache < 8 KB per multiprocessor
- 1.3 GHz clock rate
- Single and double-precision floating-point calculation
- 1 GB DDR3 dedicated memory

**Figure:** nvidia.com
- thread scheduler
- thread processing clusters
- atomic Tex L2
- Memory
GPU H/W
Parallel Computing Arch

- thread scheduler
- thread processing clusters
  - atomic Tex L2
  - Memory
GPU H/W
Parallel Computing Arch

- thread scheduler
- thread processing clusters
- atomic Tex L2

Memory
GPU H/W
Parallel Computing Arch

- thread scheduler
- thread processing clusters
- atomic Tex L2
- Memory
- **Hardware-based**
  - Manages scheduling threads across thread processing clusters
  - Nearly 100% utilization: If a thread is waiting for memory access, the scheduler can perform a zero-cost, immediate context switch to another thread
  - Up to 30,720 threads on the GPU
Hardware-based

Manages scheduling threads across thread processing clusters

Nearly 100% utilization: If a thread is waiting for memory access, the scheduler can perform a zero-cost, immediate context switch to another thread

Up to 30,720 threads on the GPU
Hardware-based
Manages scheduling threads across thread processing clusters
Nearly 100% utilization: If a thread is waiting for memory access, the scheduler can perform a zero-cost, immediate context switch to another thread
Up to 30,720 threads on the GPU
GPU H/W
thread scheduler

- Hardware-based
- Manages scheduling threads across thread processing clusters
- Nearly 100% utilization: If a thread is waiting for memory access, the scheduler can perform a zero-cost, immediate context switch to another thread
- Up to 30,720 threads on the GPU
GPU H/W
thread proc cluster

- TF - texture filtering
- IU - instruction unit

Figure: nvidia.com
GPU H/W
thread proc cluster

- TF - texture filtering
- IU - instruction unit
GPU H/W
thread proc cluster

- TF - texture filtering
- IU - instruction unit

Figure: nvidia.com

Stephen V. Providence Ph.D.  High Performance Computing Modernization Program
Level 2 Cache

- Shared by all thread processing clusters
- Atomic
  - Ability to perform read-modify-write operations to memory
  - Allows granular access to memory locations
  - Provides parallel reductions and parallel data structure management
Level 2 Cache
- Shared by all thread processing clusters
- Atomic
  - Ability to perform read-modify-write operations to memory
  - Allows granular access to memory locations
  - Provides parallel reductions and parallel data structure management
Level 2 Cache
Shared by all thread processing clusters
Atomic
- Ability to perform read-modify-write operations to memory
- Allows granular access to memory locations
- Provides parallel reductions and parallel data structure management
Level 2 Cache

Shared by all thread processing clusters

Atomic

- Ability to perform read-modify-write operations to memory
  - Allows granular access to memory locations
  - Provides parallel reductions and parallel data structure management
Level 2 Cache
- Shared by all thread processing clusters
- Atomic
  - Ability to perform read-modify-write operations to memory
  - Allows granular access to memory locations
  - Provides parallel reductions and parallel data structure management
- Level 2 Cache
- Shared by all thread processing clusters
- Atomic
  - Ability to perform read-modify-write operations to memory
  - Allows granular access to memory locations
  - Provides parallel reductions and parallel data structure management
Dynamic power management

Power consumption is based on utilization

- Idle/2D power mode: 25 W
- Blu-ray DVD playback mode: 35 W
- Full 3D performance mode: worst case 236 W
- HybridPower mode: 0 W

On an nForce motherboard, when not performing, the GPU can be powered off and computation can be diverted to the motherboard GPU (mGPU)
Dynamic power management

Power consumption is based on utilization

- Idle/2D power mode: 25 W
- Blu-ray DVD playback mode: 35 W
- Full 3D performance mode: worst case 236 W
- HybridPower mode: 0 W

On an nForce motherboard, when not performing, the GPU can be powered off and computation can be diverted to the motherboard GPU (mGPU)
Dynamic power management

Power consumption is based on utilization

- Idle/2D power mode: 25 W
- Blu-ray DVD playback mode: 35 W
- Full 3D performance mode: worst case 236 W
- HybridPower mode: 0 W

On an nForce motherboard, when not performing, the GPU can be powered off and computation can be diverted to the motherboard GPU (mGPU)
Dynamic power management

Power consumption is based on utilization

- Idle/2D power mode: 25 W
- Blu-ray DVD playback mode: 35 W
- Full 3D performance mode: worst case 236 W
- HybridPower mode: 0 W

On an nForce motherboard, when not performing, the GPU can be powered off and computation can be diverted to the motherboard GPU (mGPU)
Dynamic power management

Power consumption is based on utilization

- Idle/2D power mode: 25 W
- Blu-ray DVD playback mode: 35 W
- Full 3D performance mode: worst case 236 W

HybridPower mode: 0 W

On an nForce motherboard, when not performing, the GPU can be powered off and computation can be diverted to the motherboard GPU (mGPU)
Dynamic power management

- Power consumption is based on utilization
  - Idle/2D power mode: 25 W
  - Blu-ray DVD playback mode: 35 W
  - Full 3D performance mode: worst case 236 W
  - HybridPower mode: 0 W

- On an nForce motherboard, when not performing, the GPU can be powered off and computation can be diverted to the motherboard GPU (mGPU)
10 Thread Processing Clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- RPO - raster operation processors (for graphics)
- 1024 MB frame buffer for displaying images
- Texture (L2) Cache
10 Thread Processing Clusters (TPC)
3 multiprocessors per TPC
- 8 cores per multiprocessor
- RPO - raster operation processors (for graphics)
- 1024 MB frame buffer for displaying images
- Texture (L2) Cache
GPU H/W
240 core GPU

- 10 Thread Processing Clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- RPO - raster operation processors (for graphics)
- 1024 MB frame buffer for displaying images
- Texture (L2) Cache
10 Thread Processing Clusters (TPC)
3 multiprocessors per TPC
8 cores per multiprocessor
RPO - raster operation processors (for graphics)
1024 MB frame buffer for displaying images
Texture (L2) Cache
- 10 Thread Processing Clusters (TPC)
- 3 multiprocessors per TPC
- 8 cores per multiprocessor
- RPO - raster operation processors (for graphics)
- 1024 MB frame buffer for displaying images
- Texture (L2) Cache
10 Thread Processing Clusters (TPC)
3 multiprocessors per TPC
8 cores per multiprocessor
RPO - raster operation processors (for graphics)
1024 MB frame buffer for displaying images
Texture (L2) Cache
GPU H/W
240 core GPU image

Figure: nvidia.com
Programming Model
Past & Present

PAST
- The GPU was intended for graphics only, not general purpose computing.
- The programmer needed to rewrite the program in a graphics language, such as OpenGL
- Complicated

PRESENT
- NVIDIA developed CUDA, a language for general purpose GPU computing
- Simple
Programming Model
Past & Present

PAST
- The GPU was intended for graphics only, not general purpose computing.
- The programmer needed to rewrite the program in a graphics language, such as OpenGL
  - Complicated

PRESENT
- NVIDIA developed CUDA, a language for general purpose GPU computing
  - Simple
Programming Model
Past & Present

● PAST
  ● The GPU was intended for graphics only, not general purpose computing.
  ● The programmer needed to rewrite the program in a graphics language, such as OpenGL
  ● Complicated

● PRESENT
  ● NVIDIA developed CUDA, a language for general purpose GPU computing
  ● Simple
Programming Model
Past & Present

PAST
- The GPU was intended for graphics only, not general purpose computing.
- The programmer needed to rewrite the program in a graphics language, such as OpenGL
- Complicated

PRESENT
- NVIDIA developed CUDA, a language for general purpose GPU computing
- Simple
Compute Unified Device Architecture

- Extension of the C language
- Used to control the device
- The programmer specifies CPU and GPU functions
  - The host code can be C++
  - Device code may only be C
- The programmer specifies thread layout
- Compute Unified Device Architecture
- Extension of the C language
  - Used to control the device
  - The programmer specifies CPU and GPU functions
    - The host code can be C++
    - Device code may only be C
  - The programmer specifies thread layout
Compute Unified Device Architecture
Extension of the C language
Used to control the device
- The programmer specifies CPU and GPU functions
  - The host code can be C++
  - Device code may only be C
- The programmer specifies thread layout
Programming Model
CUDA

- Compute Unified Device Architecture
- Extension of the C language
- Used to control the device
- The programmer specifies CPU and GPU functions
  - The host code can be C++
  - Device code may only be C
- The programmer specifies thread layout
Programming Model
CUDA

- Compute Unified Device Architecture
- Extension of the C language
- Used to control the device
- The programmer specifies CPU and GPU functions
  - The host code can be C++
  - Device code may only be C
- The programmer specifies thread layout
- Compute Unified Device Architecture
- Extension of the C language
- Used to control the device
- The programmer specifies CPU and GPU functions
  - The host code can be C++
  - Device code may only be C
- The programmer specifies thread layout
Compute Unified Device Architecture
- Extension of the C language
- Used to control the device
- The programmer specifies CPU and GPU functions
  - The host code can be C++
  - Device code may only be C
- The programmer specifies thread layout
- Threads are organized into blocks.
- Blocks are organized into a grid.
- A multiprocessor executes one block at a time.
- A warp is the set of threads executed in parallel.
- 32 threads in a warp
Threads are organized into blocks.
Blocks are organized into a grid.
- A multiprocessor executes one block at a time.
- A warp is the set of threads executed in parallel.
- 32 threads in a warp
• Threads are organized into blocks.
• Blocks are organized into a grid.
• A multiprocessor executes one block at a time.
• A warp is the set of threads executed in parallel.
• 32 threads in a warp
Threads are organized into blocks.
Blocks are organized into a grid.
A multiprocessor executes one block at a time.
A warp is the set of threads executed in parallel.
32 threads in a warp
Programing Model

thread layout

- Threads are organized into blocks.
- Blocks are organized into a grid.
- A multiprocessor executes one block at a time.
- A warp is the set of threads executed in parallel.
- 32 threads in a warp
Programming Model
thread layout

Figure: nvidia.com
• GPU and CPU execute different types of code.
• CPU runs the main program, sending tasks to the GPU in the form of kernel functions
• Multiple kernel functions may be declared and called.
• Only one kernel may be called at a time.
GPU and CPU execute different types of code.

- CPU runs the main program, sending tasks to the GPU in the form of kernel functions
- Multiple kernel functions may be declared and called.
- Only one kernel may be called at a time.
GPU and CPU execute different types of code.
- CPU runs the main program, sending tasks to the GPU in the form of kernel functions
- Multiple kernel functions may be declared and called.
- Only one kernel may be called at a time.
GPU and CPU execute different types of code.
- CPU runs the main program, sending tasks to the GPU in the form of kernel functions
- Multiple kernel functions may be declared and called.
- Only one kernel may be called at a time.
Programming Model

hetero comp

C Program
Sequential
Execution

Serial code

Parallel kernel
Kernel0<<0>>();

Parallel kernel
Kernel1<<0>>();

Device

Grid 0

Block (0, 0)  Block (1, 0)  Block (2, 0)

Block (0, 1)  Block (1, 1)  Block (2, 1)

Host

Grid 1

Block (0, 0)  Block (1, 0)

Block (0, 1)  Block (1, 1)

Host

Block (0, 2)  Block (1, 2)

Figure: nvidia.com
Programming Model
GPU vs. CPU

CPU C program

```c
void add_matrix_cpu
    (float *a, float *b, float *c, int N)
{
    int i, j, index;
    for (i=0;i<N;i++) {
        for (j=0;j<N;j++) {
            index = i+j*N;
            c[index] = a[index] + b[index];
        }
    }
}

void main()
{
    ....
    add_matrix(a,b,c,N);
}
```

CUDA C program

```c
__global__ void add_matrix_gpu
    (float *a, float *b, float *c, int N)
{
    int i=blockIdx.x*blockDim.x+threadIdx.x;
    int j=blockIdx.y*blockDim.y+threadIdx.y;
    int index =i+j*N;
    if( i < N && j < N) c[index] = a[index] + b[index];
}

void main()
{
    dim3 dimBlock (blocksize,blocksize);
    dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
    add_matrix_gpu<<<<<dimGrid,dimBlock>>>(a,b,c,N);
}
```

Figure: nvidia.com
Conclusion

- SIMD causes some problems
- GPU computing is a good choice for fine-grained data-parallel programs with limited communication
- GPU computing is not so good for coarse-grained programs with a lot of communication
- The GPU has become a co-processor to the CPU
Conclusion

- SIMD causes some problems
- GPU computing is a good choice for fine-grained data-parallel programs with limited communication
- GPU computing is not so good for coarse-grained programs with a lot of communication
- The GPU has become a co-processor to the CPU
Conclusion

- SIMD causes some problems
- GPU computing is a good choice for fine-grained data-parallel programs with limited communication
- GPU computing is not so good for coarse-grained programs with a lot of communication
- The GPU has become a co-processor to the CPU
Conclusion

- SIMD causes some problems
- GPU computing is a good choice for fine-grained data-parallel programs with limited communication
- GPU computing is not so good for coarse-grained programs with a lot of communication
- The GPU has become a co-processor to the CPU
Michael J. Quinn,  
Parallel Programming in C with MPI and OpenMP  
McGraw-Hill, 2004  

J. Sanders, E. Kandrot,  
CUDA By Example: An Introduction to General-Purpose GPU Programming,  
Nvidia, 2011  

Board of Trustees of the University of Illinois, 2011  
NCSA News,  
http://www.ncsa.illinois.edu/BlueWaters/systems.html  

B. Sinharoy, et al.  
IBM POWER7 Multicore Server Processor  
For Further Reading II

Jeffrey Vetter, Dick Glassbrook, Jack Dongarra, Richard Fujimoto, Thomas Schulthess, Karsten Schwan
*Keeneland - Enabling Heterogenous Computing for the Open Science Community*
Supercomputing Conference 2010, New Orleans, Louisiana

C. Zeller, Nvidia Corporation
*C. Zeller - CUDA C Basics*
Supercomputing Conference 2010, New Orleans, Louisiana