Introduction to GPGPU

Mehmet Tunçel
Ufuk Utku Truncuoğlu
Istanbul Technical University,
Informatics Institute
Outline

• GPGPU
• GPU vs. CPU
• Thread and Memory Hierarchy
• Hardware Implementation
• GPU Programming Models
GPGPU

• **GPU** - Graphics Processing Unit

• **GPGPU** - General-Purpose computing on GPU
  • First GPGPU-enabled GPU by NVIDIA was GeForce G80

• **CUDA** - Compute Unified Device Architecture
  • is a parallel computing platform and programming model implemented by the graphics processing units created by NVIDIA
GPGPU Revolution in HPC

• Titan: World’s #2 Open Science Supercomputer

18,688 Tesla K20X GPUs
27 Petaflops Peak: 90% of Performance from GPUs
17.59 Petaflops Sustained Performance on Linpack
Acceleration via GPU

Fastest Performance on Scientific Applications

Tesla K20X Speed-Up over Sandy Bridge CPUs

<table>
<thead>
<tr>
<th>Category</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material Science</td>
<td>WL-LSMS</td>
</tr>
<tr>
<td>Higher Ed</td>
<td>MATLAB (FFT)*</td>
</tr>
<tr>
<td>Physics</td>
<td>Chroma</td>
</tr>
<tr>
<td>Earth Science</td>
<td>SPECFEM3D</td>
</tr>
<tr>
<td>Molecular Dynamics</td>
<td>AMBER</td>
</tr>
</tbody>
</table>

System Config: CPU results: Dual socket E5-2687w, 3.10 GHz
GPU results: Dual socket E5-2687w + 2 Tesla K20X GPUs
*MATLAB results comparing one I7-2600K CPU vs with Tesla K20 GPU.*
**GPU Streaming Multiprocessor (SMX)**

- **Device core unit** (similar to CPU core)
- **Consists of 1**
  - 192 CUDA cores @ ~ 1 GHz each
  - 64 FP units
  - 32 Special Function Units
  - 32 load/store units dedicated for access memories
  - 65536 registers x 32 bit (256KB)
  - 64KB shared memory / L1 cache
  - 48KB Read-Only Data cache
  - 48KB Texture cache
Kepler GK110 Full Chip Block Diagram

- 15 SMX
- 2688 CUDA Cores
- Cache L2 1.5 MB
- 384-bit GDDR5
- PCI-E 2.0
GPU vs. CPU

- Hundreds of simplified computational cores working at low clock frequencies ~1 GHz (instead of 2-8 in CPU)
- Small caches
  - 192 cores share L1 (16 - 48 KB)
  - L2 shared between all cores, 1.5 MB, no L3
- GDDR 5 with high bandwidth and high latency
  - Optimized for public access
- Zero overhead thread switching (in hardware)
  => support for millions of virtual threads
Memory Latency Utilization

- Purpose: load all cores
  - Problem: memory latency

- Solution:
  - CPU: complex caches hierarchy
  - GPU: thousands of threads ready to be executed
Development Ecosystem

Applications

Libraries
Drop-in Acceleration

Directive Based
Easily Accelerate Applications

Programming Languages
Maximum Performance
Thread Hierarchy

GRID

BLOCK

THREAD
Hardware Implementation

• All threads of a single block are executed on a single multiprocessor (SMX)
• Blocks can’t switch SMX
• Allocation of blocks between multiprocessors is unpredictable
• Each SMX operates independently
Memory Hierarchy

Host

Device

MemCopy

Kernel 1

Grid 1

Block (0, 0)

Block (1, 0)

Block (2, 0)

Block (0, 1)

Block (1, 1)

Block (2, 1)

Grid 2

Block (1, 1)

Thread (0, 0)

Thread (1, 0)

Thread (2, 0)

Thread (3, 0)

Thread (4, 0)

Thread (0, 1)

Thread (1, 1)

Thread (2, 1)

Thread (3, 1)

Thread (4, 1)

MemCopy

Kernel 2

Grid

Block (0, 0)

Shared Memory

Registers

Thread (0, 0)

Thread (1, 0)

Local Memory

Global Memory

Constant Memory

Texture Memory

Block (1, 0)

Shared Memory

Registers

Thread (0, 0)

Thread (1, 0)

Local Memory

Global Memory

Constant Memory

Texture Memory
CUDA Kernels and Threads

- Parallel part of an application is executed on the device as **kernels**. The kernels are also called as “**device code**”.
- It is an entry point for code executed on GPU
  1. The host (CPU) launches **kernels** and **device** (GPU) executes them
  2. One **kernel** executed at a time
  3. The **kernel** is executed by many **threads**
  4. Thread can only access GPU memory (the data must be transferred from CPU memory to GPU one before execution!)
- CUDA threads have less overhead. They can easily be created and switching between them is fast
- CPU can create few threads but CUDA uses 1000s
GPU Programming Models

• Intra-node
  – Single GPU
  – Multi GPU
    • MPI /OpenMP

• Inter-node
  – Data transfer
References

https://developer.nvidia.com
http://www.parallel-computing.pro/