CSCI-GA.3033-004
Graphics Processing Units (GPUs): Architecture and Programming

Lecture 12: What’s Next: Software

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
This Lecture

• Computational Thinking!
• Tools
• Libraries
• State-of-the-art and future applications
Computational Thinking is arguably the most important aspect of parallel Application development!
J. Wing Communications of the ACM, 49(3), 2006

What is it?
Decomposing a domain problem into well-defined, coordinated work units that can Each be realized with different numerical methods and well-known algorithms.
Why Do We Need Parallel Computing in the First place?

To solve a given problem in less time

To solve bigger problems

To achieve better solutions for a given problem and a given amount of time

Increased Speed!
Applications that are good candidates for parallel computing:

- Involve large problem sizes
- Involve high modeling complexity

Formulating the problem is crucial!!

The problem must be formulated in a such a way that it can be decomposed into subproblems that can be executed at the same time.
The Process of Parallel Programming

• Problem decomposition
• Algorithm selection
• Implementation in a language
• Performance tuning

This is what we have been doing till now!
Problem Decomposition

Identify the work to be performed by each unit of parallel execution → thread in CUDA
Problem Decomposition: Thread Arrangement

Example: Electrostatic Map Problem

Atom-centric: each thread responsible for calculating the effect of one atom on all grid points $\rightarrow$ Scatter

Grid-centric: each thread calculates the effect of all atoms on a grid point $\rightarrow$ Gather
Problem Decomposition

• Picking the best thread arrangement requires the understanding of the underlying hardware.

• A real application consists of several modules that work together
  – Amount of work per module can vary dramatically
  – You need to decide if a module is worth implementing in CUDA

• Amdahl’s law
Algorithm Selection

• An algorithm must exhibit three essential properties:
  – definiteness = no ambiguity
  – effective computability = each step can be carried by a computer
  – finiteness = guaranteed to terminate

• When comparing several algorithms, take the following factors into account:
  – Steps of computation
  – Degree of parallel execution
  – Numerical stability
  – Memory bandwidth
Skills needed to go from: Parallel Programmer to: Computational Thinker

- Computer Architecture
- Programming models and compilers
- Algorithm techniques: (e.g. tiling)
- Domain knowledge
Tools

https://developer.nvidia.com/

• Almost all tools are available for free
• Many of them are cross-platform
CUDA-GDB

• Compile your program with:
  
  `nvcc -G -g yourprog.cu -o anyname`

  – The -g option tells nvcc to generate debugging information for the host code
  – The -G option generates debugging information for the CUDA kernels

• Start with: `cuda-gdb anyname`

• You will get the prompt: `$(cuda-gdb)`
CUDA-GDB

- $(cuda-gdb) \ l 60$
  - lists the code around line 60
- $(cuda-gdb) \ b 60$
  - puts a breakpoint at line 60
- $(cuda-gdb) \ b \ mykernel$
  - puts a breakpoint at kernel mykernel
- $(cuda-gdb) \ r$
  - runs the program
CUDA-GDB

- $(cuda-gdb)\ n$
  - runs the next line of the program
  - Important: The debugger supports stepping GPU code at the granularity of a warp.
- $(cuda-gdb)\ c$
  - continues execution till the next breakpoint
- $(cuda-gdb)\ p\ \ *v\_d@10$
  - Print the first 10 elements of array $v\_d$
- $(cuda-gdb)\ info\ cuda\ threads$
  - print the thread and block numbers at current kernel in the form $\langle\langle BX,BY),(TX,TY,TZ)\rangle\rangle$
- $(cuda-gdb)\ thread\ \langle\langle(31),(255)\rangle\rangle$
  - switch to that thread in case you want to print info about it.
- $(cuda-gdb)\ quit$
PGI CUDA C/C++ for x86

- compiler for x86 platforms
- allows developers using CUDA to compile and optimize their CUDA applications to run on x86-based workstations
- with or without an NVIDIA GPU
- When no NVIDIA GPU exists, PGI CUDA C uses multiple cores and the streaming SIMD capabilities of Intel and AMD CPUs
CUDA Libraries

• CUDA provides a set of very useful libraries.
• This increases the programmer productivity.

Libraries

- CUBLAS
- CUFFT
- MAGMA
- CULA
- Thrust
- ...

CUDA Libraries: CUBLAS

- Cuda Based Linear Algebra Subroutines
  - Saxpy, conjugate gradient, linear solvers, ...
- Single GPU or Multiple GPUs
- Support CUDA Stream
- Basic preparation
  - Install CUDA Toolkit
  - Include cublas_v2.h
  - Link cublas.lib
CUDA Libraries: CUBLAS

• Some basic tips
  – Every CUBLAS function needs a handle
  – The CUBLAS function must be written between `cublasCreate()` and `cublasDestory()`
  – Every CUBLAS function returns a `cublasStatus_t` to report the state of execution.
  – Column-major storage
CUDA Libraries: CUBLAS

- **Single precision data:**
  - Level 1 (vector-vector $O(N)$)
  - Level 2 (matrix-vector $O(N^2)$)
  - Level 3 (matrix-matrix $O(N^3)$)

- **Complex single precision data:**
  - Level 1
  - CGEMM

- **Double precision data:**
  - Level 1: DASUM, DAXPY, DCOPY, DDOT, DNRM2, DROT, DROTM, DSCAL, DSWAP, ISAMAX, IDAMIN
  - Level 2: DGEMV, DGER, DSYR, DTRSV
  - Level 3: ZGEMM, DGEMM, DTRSM, DTRMM, DSYMM, DSYRK, DSYR2K
CUDA Libraries: CUFFT

- Cuda Based Fast Fourier Transform Library.
- The FFT is a divide-and-conquer algorithm
- Computes FFT on Nvidia CUDA
- 1D, 2D, and 3D
- The CUFFT library is freely available as part of the CUDA Toolkit
- `#include<cufft.h>"
For sparse matrices

a C++ template library for CUDA

based on the Standard Template Library
GPUs in Real Life
"Harvard Middle Eastern Studies student Todd Mostak's first tangle with big data didn't go well; trying to process and map 40 million geolocated tweets from the Arab Spring uprising took days. So while taking a database course across town at MIT, he developed a massively parallel database that uses GeForce Titan GPUs to do the data processing. The system sees 70x performance increases over CPU-based systems, and can out crunch a 1000 node MapReduce cluster, in some cases. All for around $5,000 worth of hardware. Mostak plans to release the system under an open source license; you can play with a data set of 125 million tweets hosted at Harvard's WorldMap and see the millisecond response time.

Source: Slashdot, April 22nd, 2013

http://slashdot.org/story/13/04/22/2225240/harvardmit-student-creates-gpu-database-hacker-style
GPUs with DataBases

Map-D, a startup based in Cambridge, Mass., has built a high-speed GPU in-memory database and geospatial visualization tool that can track more than a billion tweets worldwide - and provide real-time interactive visual analysis of an almost boundless number of socio-economic queries.

Source:
How Can GPUs Help With Databases?

- Sorting
- Accelerating SQL

MapReduce and GPUs

• MapReduce Programming Model
  – Emerged with the development of Data-Intensive Computing
  – Pioneered by Google
  – Map()
    • Generates a large number of (key, value) pairs
  – Reduce()
    • Merges the values associated with the same key
Example: Distributed Grep

Very big data

\[\text{Split data} \rightarrow \text{grep} \rightarrow \text{matches} \]
\[\text{Split data} \rightarrow \text{grep} \rightarrow \text{matches} \]
\[\text{Split data} \rightarrow \text{grep} \rightarrow \text{matches} \]
\[\text{Split data} \rightarrow \text{grep} \rightarrow \text{matches} \]
\[cat \rightarrow \text{All matches} \]
Example: Distributed Word Count

Very big data

Split data → count → count → merge → merged count

Split data

Split data

Split data
So: Map+Reduce

- **Map:**
  - Accepts *input* key/value pair
  - Emits *intermediate* key/value pair

- **Reduce:**
  - Accepts *intermediate* key/value* pair
  - Emits *output* key/value pair

How can GPU help in this?
GPUs in Supercomputers

• Strongest supercomputers now are based on GPUs
• No shared memory → MPI is the way to go
• Nodes connected by sophisticated interconnection
• Can we combine CUDA and MPI?
Blue Waters Computing System

- **Sonexion**: 26 PBs, >1 TB/sec, 100 GB/sec
- **10/40/100 Gb Ethernet Switch**
- **IB Switch**: >1 TB/sec
- **WAN**: 120+ Gb/sec
- **Spectra Logic**: 300 PBs, 100 GB/sec
- **Sonexion**: 26 PBs
## Blue Waters and Titan Computing Systems

<table>
<thead>
<tr>
<th>System Attribute</th>
<th>NCSA Blue Waters</th>
<th>ORNL Titan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendors</td>
<td>Cray/AMD/NVIDIA Interlagos/Kepler</td>
<td>Cray/AMD/NVIDIA Interlagos/Kepler</td>
</tr>
<tr>
<td>Processors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Peak Performance (PF)</td>
<td>11.1</td>
<td>27.1</td>
</tr>
<tr>
<td>Total Peak Performance (CPU/GPU)</td>
<td>7.1/4</td>
<td>2.6/24.5</td>
</tr>
<tr>
<td>Number of CPU Chips</td>
<td>48,352</td>
<td>18,688</td>
</tr>
<tr>
<td>Number of GPU Chips</td>
<td>3,072</td>
<td>18,688</td>
</tr>
<tr>
<td>Amount of CPU Memory (TB)</td>
<td>1511</td>
<td>584</td>
</tr>
<tr>
<td>Interconnect</td>
<td>3D Torus</td>
<td>3D Torus</td>
</tr>
<tr>
<td>Amount of On-line Disk Storage (PB)</td>
<td>26</td>
<td>13.6</td>
</tr>
<tr>
<td>Sustained Disk Transfer (TB/sec)</td>
<td>&gt;1</td>
<td>0.4-0.7</td>
</tr>
<tr>
<td>Amount of Archival Storage</td>
<td>300</td>
<td>15-30</td>
</tr>
<tr>
<td>Sustained Tape Transfer (GB/sec)</td>
<td>100</td>
<td>7</td>
</tr>
</tbody>
</table>
CUDA-based cluster

• Each node contains $N$ GPUs
MPI Model

- Many processes distributed in a cluster
- Each process computes part of the output
- Processes communicate with each other
- Processes can synchronize
MPI Sending Data

- int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
  - Buf: Initial address of send buffer (choice)
  - Count: Number of elements in send buffer (nonnegative integer)
  - Datatype: Datatype of each send buffer element (handle)
  - Dest: Rank of destination (integer)
  - Tag: Message tag (integer)
  - Comm: Communicator (handle)
MPI Receiving Data

- `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
  - **Buf**: Initial address of receive buffer (choice)
  - **Count**: Maximum number of elements in receive buffer (integer)
  - **Datatype**: Datatype of each receive buffer element (handle)
  - **Source**: Rank of source (integer)
  - **Tag**: Message tag (integer)
  - **Comm**: Communicator (handle)
  - **Status**: Status object (Status)
MPI + CUDA: The Easy Way

• Cuda and MPI can be considered separate entities
  – CUDA handles parallelization on GPU
  – MPI handles parallelization over nodes

• Use one MPI process per GPU and accelerate the computational kernels with CUDA
MPI + CUDA: The Hard Part

• Technology is moving forward quickly
  – Different compute capability generations
  – Different levels of support for GPUDirect
  – New MPI libraries with CUDA support are emerging

• Machines differ from each other
  – Number of GPUs and CPUs per node differ
    • 1 GPU per 2 processors to 8 GPUs per 2 processors
    • Selecting active GPU on multi GPU nodes

• Scalability
Scalability across GPU/CPU cluster nodes
(big hybrid supercomputers are coming)

Oak Ridge National Laboratory looks to NVIDIA “Fermi” architecture for new supercomputer

NERSC experimental GPU cluster: Dirac

EMSL experimental GPU cluster: Barracuda
GPU in Finance

• Why? Opportunity & Risk!
  – Opportunity: doing things faster $\rightarrow$ better
    $\rightarrow$ more opportunities
  – Risk: market risk analysis

• Real time

• Relative to multicore: GPU is easier
  better or cheaper!
GPU in Finance

• Example of applications:
  – Pricing
  – Trading strategy
  – Risk analysis
  – Data visualization
  – High-frequency trading
  – Data mining

• Monte Carlo → Library CUDA THRUST
Conclusions

• Computational thinking is an art but a very crucial one.
• There are now wrappers of CUDA for many languages: Python, Java, C++, Matlab, ...
• GPUs are gaining territory as new applications arise!