Course Goals

• Learn to program massively parallel processors and achieve
  – High performance
  – Functionality and maintainability
  – Scalability across future generations

• Technical subjects
  – Parallel programming basics
  – Principles and patterns of parallel algorithms
  – Programming API, tools and techniques
  – Processor architecture features and constraints
  – Killer apps

People

Professor:

Steve Lumetta
lumetta@illinois.edu
use ECE408 to start your e-mail subject line

Web Resources

• web page: http://lumetta.web.engr.illinois.edu/408-S19/
  – Announcements and handouts
  – Links to lecture slides/recording from 2015
• additional resources (linked from web page):
  http://ece408.hwu-server2.crhc.illinois.edu/
• web board discussions in Piazza
  – Channel for electronic announcements
  – Forum for Q&A – I will read the board, and your classmates often have answers
• Blackboard - grades
Grading

• Exams: 40%
  – Exam 1: 20% -- 9 April 6:30 to 9:30 p.m.
  – Exam 2: 20% -- 23 May 6:30 to 9:30 p.m.

• Labs (Machine Problems): 35%
  – Passing Datasets 90%
  – Reasonable-looking answers to questions

• Project: 25%
  – Demo/Functionality/Coding Style: 50%
  – Performance Ranking with full functionality: 50%
  – Detailed Rubric will be posted

Academic Honesty

• You are allowed and encouraged to discuss assignments with other students in the class. Getting verbal advice/help from people who’ve already taken the course is also fine.

• Any reference to assignments from previous terms or web postings is unacceptable

• Any copying of non-trivial code is unacceptable
  – Non-trivial = more than a line or so
  – Copying includes reading someone else’s code and then going off to write your own.

Academic Honesty (cont.)

• Giving/receiving help on an exam is unacceptable

• Penalties for academic dishonesty:
  – Zero on the assignment/exam for the first occasion
  – Automatic failure of the course for repeat offenses

Text/Notes


2. NVIDIA, *NVidia CUDA C Programming Guide*, version 7.5 or later (reference book)
Tentative Schedule

• “Week 1”:
  – Sat: Lecture 1: Introduction
  – Tues: Lecture 2: CUDA Intro

• “Week 2”:
  – Thurs: Lecture 3: Data Parallelism Model
  – Sat: Lecture 4: CUDA Memory Model
  – Due: MP-0, installation, test account, MP-1, vector addition

• Week 3:
  – Tu: Lecture 5: CUDA Memory Model
  – Th: Lecture 6: Performance Considerations
  – Due: MP-2, simple matrix multiplication

A major paradigm shift

- In the 20th Century, we were able to understand, design, and manufacture what we can measure
  - Physical instruments and computing systems allowed us to see farther, capture more, communicate better, understand natural processes, control artificial processes…

Examples of Paradigm Shift

20th Century
- Small mask patterns
- Electronic microscope and Crystallography with computational image processing
- Anatomic imaging with computational image processing
- Teleconference
- GPS

21st Century
- Optical proximity correction
- Computational microscope with initial conditions from Crystallography
- Metabolic imaging sees disease before visible anatomic change
- Tele-immersion
- Self-driving cars

A major paradigm shift

- In the 21st Century, we are able to understand, design, and create what we can compute
  - Computational models are allowing us to see even farther, going back and forth in time, learn better, test hypothesis that cannot be verified any other way, create safe artificial processes…
POST-DENNARD TECHNOLOGY PIVOT – PARALLELISM AND HETEROGENEITY

Dennard Scaling of MOS Devices
- In this ideal scaling, as \( L \rightarrow \alpha^*L \)
  - \( V_{DD} \rightarrow \alpha^*V_{DD} \)
  - \( C \rightarrow \alpha^*C \)
  - \( i \rightarrow \alpha^*i \)
- Delay = \( CV_{dd}/I \) scales by \( \alpha \), so \( f \rightarrow 1/\alpha \)
- Power for each transistor is \( CV^2f \) and scales by \( \alpha^2 \)
  - keeping total power constant for same chip area

Frequency Scaled Too Fast 1993-2003

Total Processor Power Increased
(super-scaling of frequency and chip size)
Post-Dennard Pivoting

- Multiple cores with more moderate clock frequencies
- Heavy use of vector execution
- Employ both latency-oriented and throughput-oriented cores
- 3D packaging for more memory bandwidth

Blue Waters Computing System
Operational at Illinois since 3/2013

Sonexion: 26 PBs
WAN
Spectra Logic: 300 PBs

Qualcomm SoC for Mobile

Cray XK7 Compute Node

XM7 Compute Node Characteristics

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Series 6200 (Interlagos)</td>
<td></td>
</tr>
<tr>
<td>NVIDIA Kepler</td>
<td>Host Memory: 32 GB</td>
</tr>
<tr>
<td></td>
<td>1600 MT/s DDR3</td>
</tr>
<tr>
<td>NVIDIA Tesla X2090 Memory</td>
<td>6GB GDDR5 capacity</td>
</tr>
<tr>
<td>Gemini High Speed Interconnect</td>
<td>Keplers in final installation</td>
</tr>
</tbody>
</table>
CPUs: Latency Oriented Design

- High clock frequency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency
- Powerful ALU
  - Reduced operation latency

GPU: Throughput Oriented Design

- Moderate clock frequency
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies

Winning Strategies Use Both CPU and GPU

- CPUs for sequential parts where latency hurts
  - CPUs can be 10+X faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
  - GPUs can be 10+X faster than CPUs for parallel code

Heterogeneous Parallel Computing Applications

- Financial Analysis
- Scientific Simulation
- Engineering Simulation
- Data Intensive Analytics
- Medical Imaging
- Digital Audio Processing
- Digital Video Processing
- Computer Vision
- Machine Learning
- Electronic Design Automation
- Biomedical Informatics
- Statistical Modeling
- Ray Tracing Rendering
- Interactive Physics
- Numerical Methods
Parallel Programming Work Flow

- Identify compute intensive parts of an application
- Adopt/create scalable algorithms
- Optimize data arrangements to maximize locality
- Performance Tuning
- Pay attention to code portability, scalability, and maintainability

Algorithm Complexity and Data Scalability

A Real Example of Data Scalability
Particle-Mesh Algorithms
Load Balance

- The total amount of time to complete a parallel job is limited by the thread that takes the longest to finish

Global Memory Bandwidth

- Ideal
- Reality

Conflicting Data Accesses Cause Serialization and Delays

- Massively parallel execution cannot afford serialization
- Contentions in accessing critical data causes serialization
What is the stake?

• Scalable and portable software lasts through many hardware generations

*Scalable algorithms and libraries can be the best legacy we can leave behind from this era*

ANY MORE QUESTIONS?