CS6963

Parallel Programming for Graphics Processing Units (GPUs)

Lecture 1: Introduction

Course Information

CS6963: Parallel Programming for GPUs, MW 10:45-12:05, MEB 3105

- Website: http://www.eng.utah.edu/~cs6963/
- Professor:
  Mary Hall
  MEB 3466, mhall@cs.utah.edu, 5-1039
  Office hours: 12:20-1:20 PM, Mondays
- Teaching Assistant:
  Sriram Aananthakrishnan
  MEB 3157, sriram@cs.utah.edu
  Office hours: 2-3PM, Thursdays

Lectures (slides and MP3) will be posted on website.

Source Materials for Today's Lecture

- Wen-mei Hwu (UIUC) and David Kirk (NVIDIA)
  http://courses.ece.uiuc.edu/ece498/al1/
- Jim Demmel (UCB) and Kathy Yelick (UCB, NERSC)
  http://www.eecs.berkeley.edu/~yelick/cs267_sp07/lectures
- NVIDIA:
  http://www.nvidia.com
- Others as noted

Course Objectives

- Learn how to program "graphics" processors for general-purpose multi-core computing applications
  - Learn how to think in parallel and write correct parallel programs
  - Achieve performance and scalability through understanding of architecture and software mapping
- Significant hands-on programming experience
  - Develop real applications on real hardware
- Discuss the current parallel computing context
  - What are the drivers that make this course timely
  - Contemporary programming models and architectures, and where is the field going

Grading Criteria

- Homeworks and mini-projects: 25%
- Midterm test: 15%
- Project proposal: 10%
- Project design review: 10%
- Project presentation/demo 15%
- Project final report 20%
- Class participation 5%

Primary Grade: Team Projects

- Some logistical issues:
  - 2-3 person teams
  - Projects will start in late February
- Three parts:
  - (1) Proposal; (2) Design review; (3) Final report and demo
- Application code:
  - Most students will work on MPM, a particle-in-cell code.
  - Alternative applications must be approved by me (start early).

Collaboration Policy

- I encourage discussion and exchange of information between students.
- But the final work must be your own.
  - Do not copy code, tests, assignments or written reports.
  - Do not allow others to copy your code, tests, assignments or written reports.

Lab Information

Primary lab

- "lab6" in WEB 130
- Windows machines
- Accounts are supposed to be set up for all who were registered as of Friday
- Contact opers@eng.utah.edu with questions

Secondary

- Until we get to timing experiments, assignments can be completed on any machine running CUDA 2.0 (Linux, Windows, MAC OS)

Tertiary

- Tesla S1070 system expected soon

Text and Notes

1. NVidia, *CUDA Programmng Guide,* available from http://www.nvidia.com/object/cuda_develop.html for CUDA 2.0 and Windows, Linux or MAC OS.
2. [Recommended] M. Pharr (ed.), *GPU Gems 2* – Programming Techniques for High Performance Graphics and General-Purpose Computation, Addison Wesley, 2005. http://http.developer.nvidia.com/GPUGems2/gpugems2_part01.html
3. [Additional] Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing, 2nd Ed.* (Addison-Wesley, 2003).
4. Additional readings associated with lectures.

Schedule: A Few Make-up Classes

A few make-up classes needed due to my travel

Time slot: Friday, 10:45-12:05, MEB 3105

Dates: February 20, March 13, April 3, April 24

Today's Lecture

- Overview of course (done)
- Important problems require powerful computers …
  - … and powerful computers must be parallel.
  - Increasing importance of educating ***parallel programmers*** (you!)
- Why graphics processors?
  - Opportunities and limitations
- Developing ***high-performance*** parallel applications
  - An optimization perspective

Parallel and Distributed Computing

- Limited to supercomputers?
  - No! Everywhere!
- Scientific applications?
  - These are still important, but also many ***new*** commercial applications and ***new*** consumer applications are going to emerge.
- Programming tools adequate and established?
  - No! Many ***new*** research challenges

↑
My Research Area

Why we need powerful computers

Scientific Simulation: The Third Pillar of Science

- Traditional scientific and engineering paradigm:
  1) Do theory or paper design.
  2) Perform experiments or build system.
- Limitations:
  - Too difficult -- build large wind tunnels.
  - Too expensive -- build a throw-away passenger jet.
  - Too slow -- wait for climate or galactic evolution.
  - Too dangerous -- weapons, drug design, climate experimentation.
- Computational science paradigm:
  3) Use high performance computer systems to simulate the phenomenon
     - Base on known physical laws and efficient numerical methods.

Slide source: Jim Demmel, UC Berkeley

The quest for increasingly more powerful machines

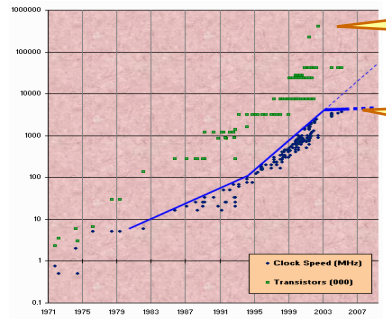
- Scientific simulation will continue to push on system requirements:
  - To increase the precision of the result
  - To get to an answer sooner (e.g., climate modeling, disaster modeling)
- The U.S. will continue to acquire systems of increasing scale
  - For the above reasons
  - And to maintain competitiveness

A Similar Phenomenon in Commodity Systems

- More capabilities in software
- Integration across software
- Faster response
- More realistic graphics
- …

Why powerful computers must be parallel

Technology Trends: Moore's Law

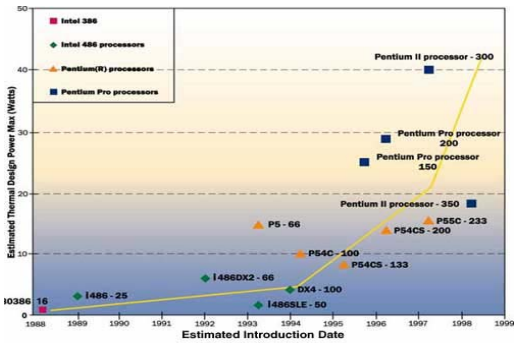


Transistor count still rising

Clock speed flattening sharply

Slide from Maurice Herlihy

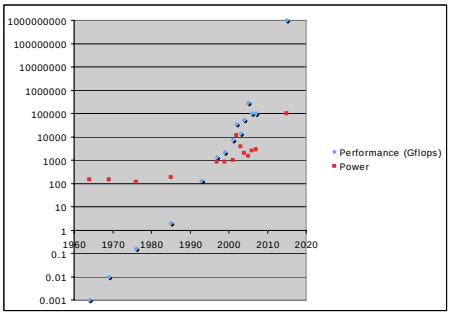
Techology Trends: Power Issues



From www.electronics-cooling.com/.../jan00_a2f2.jpg

Power Perspective

GigaFlop/s



MegaWatts

Slide source: Bob Lucas

The Multi-Core Paradigm Shift

What to do with all these transistors?

- Key ideas:
  - Movement away from increasingly complex processor design and faster clocks
  - Replicated functionality **(i.e., parallel)** is simpler to design
  - Resources more efficiently utilized
  - Huge power management advantages

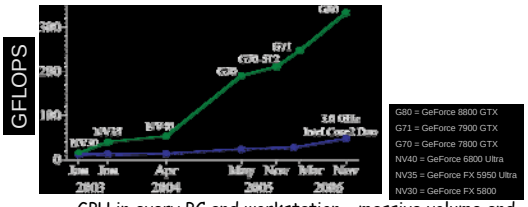
All Computers are Parallel Computers.

Who Should Care About Performance Now?

- Everyone! (Almost)
  - Sequential programs will not get faster
    - If individual processors are simplified and compete for shared resources.
    - And forget about adding new capabilities to the software!
  - Parallel programs will also get slower
    - Quest for coarse-grain parallelism at odds with smaller storage structures and limited bandwidth.
  - Managing locality even more important than parallelism!
    - Hierarchies of storage and compute structures
- Small concession: some programs are nevertheless fast enough

Why Massively Parallel Processor

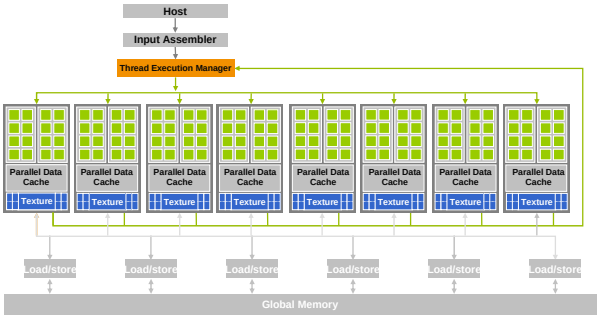
- A quiet revolution and potential build-up
  - Calculation: 367 GFLOPS vs. 32 GFLOPS
  - Memory Bandwidth: 86.4 GB/s vs. 8.4 GB/s
  - Until last year, programmed through graphics API



  - GPU in every PC and workstation – massive volume and potential impact

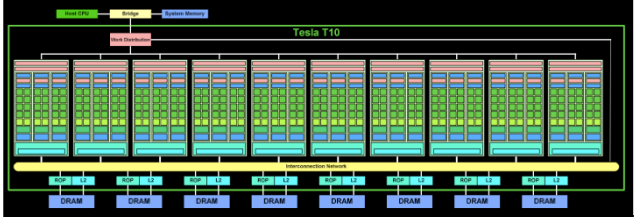
GeForce 8800

16 highly threaded SM's, >128 FPU's, 367 GFLOPS, 768 MB DRAM, 86.4 GB/S Mem BW, 4GB/S BW to CPU



Tesla 10-Series Architecture

- Massively parallel general computing architecture
- 30 Streaming multiprocessors @ 1.45 GHz with 4.0 GB of RAM
  - 1 TFLOPS single precision (IEEE 754 floating point)
  - 87 GFLOPS double precision



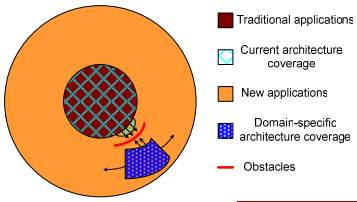
Concept of GPGPU (General-Purpose Computing on GPUs)

- Idea:
  - Potential for very high performance at low cost
  - Architecture well suited for certain kinds of parallel applications ***(data parallel)***
  - Demonstrations of ***30-100X*** speedup over CPU
- Early challenges:
  - Architectures very customized to graphics problems (e.g., vertex and fragment processors)
  - Programmed using graphics-specific programming models or libraries
- Recent trends:
  - Some convergence between commodity and GPUs and their associated parallel programming models

See http://gpgpu.org

Stretching Traditional Architectures

- Traditional parallel architectures cover some super-applications
  - DSP, GPU, network apps, Scientific, Transactions
- The game is to grow mainstream architectures "out" or domain-specific architectures "in"
  - CUDA is latter



The fastest computer in the world today

What is its name?	RoadRunner
Where is it located?	Los Alamos National Laboratory
How many processors does it have?	~19,000 processor chips (~129,600 "processors")
What kind of processors?	AMD Opterons and IBM Cell/BE (in Playstations)
How fast is it?	1.105 Petaflop/second One quadrilion operations/s 1×10^{16}

See http://www.top500.org

Parallel Programming Complexity

An Analogy to Preparing Thanksgiving Dinner

- Enough parallelism? (Amdahl's Law)
  - Suppose you want to just serve turkey
- Granularity
  - How frequently must each assistant report to the chef
    - After each stroke of a knife? Each step of a recipe? Each dish completed?
- ***All of these things makes parallel programming even harder than sequential programming.***
- Load balance
  - Each assistant gets a dish? Preparing stuffing vs. cooking green beans?
- Coordination and Synchronization
  - Person chopping onions for stuffing can also supply green beans
  - Start pie after turkey is out of the oven

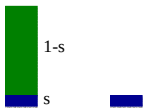
Finding Enough Parallelism

- Suppose only part of an application seems parallel
- Amdahl's law
  - let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable
  - P = number of processors

Speedup(P) = Time(1)/Time(P)

<= 1/(s + (1-s)/P)

<= 1/s

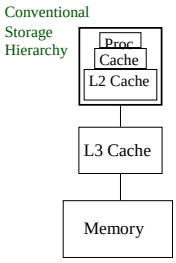


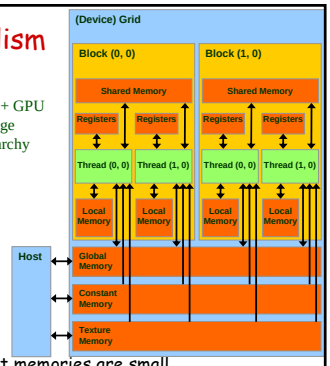
- Even if the parallel part speeds up perfectly, performance is limited by the sequential part

Overhead of Parallelism

- Given enough parallel work, this is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (I.e. large granularity), but not so large that there is not enough parallel work

Locality and Parallelism





Courtesy NVIDIA

- Large memories are slow, fast memories are small
- Storage hierarchies are large and fast on average
- Algorithm should do most work on nearby data

Load Imbalance

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Examples of the latter
  - different control flow paths on different tasks
  - adapting to "interesting parts of a domain"
  - tree-structured computations
  - fundamentally unstructured problems
- Algorithm needs to balance load

Summary of Lecture

- Technology trends have caused the ***multi-core paradigm shift*** in computer architecture
  - Every computer architecture is parallel
- Parallel programming is reaching the masses
  - This course will help prepare you for the future of programming.
- We are seeing some convergence of graphics and general-purpose computing
  - Graphics processors can achieve high performance for more general-purpose applications
- GPGPU computing
  - Heterogeneous, suitable for data-parallel applications

Next Time

- Immersion!
  - Introduction to CUDA