CME213/ME339
Lecture 1

Eric Darve
Erich Elsen
Austin Gibbons

Department of Mechanical Engineering
Institute for Computational and Mathematical Engineering
Stanford University

Spring 2012

Organization of the class

- Instructor: Eric Darve, darve@stanford.edu, 650 918 6407; office hours: Tuesday 2PM-4PM
- Instructor: Erich Elsen, ekelsen@gmail.com
- Teaching Assistant: Austin Gibbons, gibbons4@stanford.edu
- Forum/class site: piazza.com
- Website: cme213.stanford.edu (courseware web site)

Parallel computing: 1, 2, 3

1. **CUDA** and NVIDIA GPUs (graphics processing units): hundreds of cores operate concurrently; memory is shared; programmers write thread-level parallel code; single instruction controls multiple processing elements.
2. **OpenMP:** *shared-memory model* or unified memory address space (physically, may or may not be a shared memory); number of cores is typically 4–32.
3. **MPI** (message passing interface): *distributed-memory model* (again the physical layout of memory may or may not be distributed); typically corresponds to multiple computers connected by a fast network; communication happens by exchanging messages; number of nodes can be in the hundreds or more.

Grading

- 5 Homeworks: 3 CUDA, 1 openMP, 1 MPI. 14% each = 70 %
- One final project: 30%

Final project

- The project will be focussed on CUDA. The goal will be to produce an efficient code to solve a given problem.
- Content of project: analysis of problem, validation of code, optimization and benchmark of code
- Evaluation: correctness, benchmark tests, analysis of performance

Final Project Schedule

- Starts on Wednesday May 16.
- Benchmarks submitted on Friday June 1st. Code must run correctly for all test cases. 20% of grade.
- Benchmarks submitted on Monday June 4th. 20% of grade based on correctness and performance. Based on results, oral presentations are scheduled for Friday June 8th.
- End of project on Friday June 8th. 60% of grade. Oral presentation if selected, if not a 4-page written report will be prepared.

Schedule of class

This is not a contract but the current view of the schedule. It may be changed as the quarter progresses.

- Monday April 2 – Wednesday May 2: CUDA
- Friday May 4 – Wednesday May 9: openMP
- Monday May 14 – Wednesday May 23: MPI
- Wednesday May 30 – Wednesday June 6: NVIDIA lectures

Homework Schedule

1. CUDA 1: due Wednesday April 18
2. CUDA 2: due Wednesday April 25
3. CUDA 3: due Wednesday May 9
4. openMP: due Wednesday May 16
5. MPI: due Wednesday May 23

Why parallel computing?

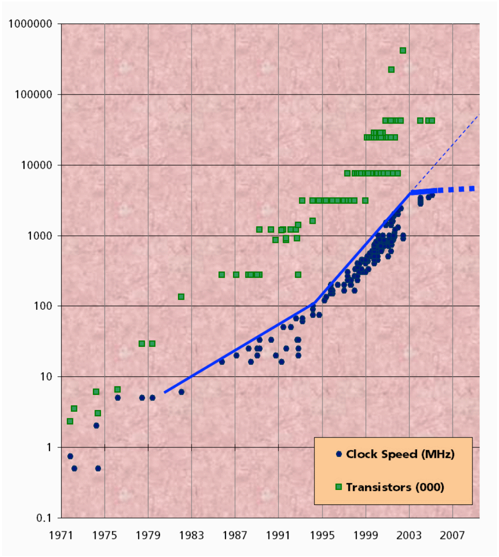
Parallel computing has existed for a long time but until recently it was a specialized area that concerned only a small fraction of engineers.

Nowadays, parallel computing is a dominant player in scientific and large scale computing.

What happened?

Until 2003, the performance improvement in processors was obtained through two technological advances:

- Moore's law: predicts exponential growth of transistors' density in micro-processors.
- Until 2003, this has translated into higher clock frequency and better performance.


1000000
100000
10000
1000
100
10
1
0.1
1971
1975
1979
1983
1987
1991
1995
1999
2003
2007
● Clock Speed (MHz)
🟩 Transistors (000)

CPU optimization during the last decades

The traditional approaches for CPU optimizations have focussed on:

- clock speed: executing faster
- execution optimization: doing more work per cycle
- cache: reducing the number of accesses to the slow main memory

This can be achieved using various techniques: pipelining, branch prediction, executing multiple instructions in the same clock cycle(s), reordering the instruction stream for out-of-order execution.

A “free” lunch

“Andy giveth, and Bill taketh away.”

Software could rely on hardware to provide performance improvements.

No matter how fast processors get, software consistently finds new ways to eat up the extra speed. Make a CPU ten times as fast, and software will usually find ten times as much to do (or, in some cases, will feel at liberty to do it ten times less efficiently).

Physical limitations

However, this performance race has now hit physical limits having to do with the size of the components (wires) in micro-processor fabrication.

Because of:

- the heat generated,
- the power-consumption,
- the current leakage,

recent processors are not faster than the previous generation. They may even be slower.

Concurrency

Despite this, performance is still improving and Moore's law will still apply for the coming years.

This is achieved through concurrency: processors and computers are becoming parallel, that is they have multiple computing units that can work at the same time to perform more operations.

Two separate trajectories are appearing at this time: multi-core and many-core.

Multicore

Multicore processors. Example: Intel Core i7. Sandy Bridge 3930K 3.8 GHz. 243 Gflops.

Note: the multicore/manycore terminology is not set in stone and its precise meaning varies.

Out-of-order, multiple-instruction issue processor implementing the full x86 instruction set. 4 to 6 cores on each processor operate in parallel. The microprocessor supports hyperthreading with two hardware threads (more parallelism with the "same" hardware). The processor is still designed to maximize the execution speed of sequential programs but parallel codes are required for peak performance.

Manycore

Manycore processors. Focuses more on the execution throughput of parallel applications.

Large number of much smaller cores.

Once again, the number of cores approximately doubles with each generation.

Current exemplar: NVIDIA Tesla M2090 graphics processing unit (GPU) with 512 cores, each of which is a heavily multithreaded, in-order, single-instruction issue processor.

Peak performance: 1,331 Gflops. Maximum number of threads: 24,576. Memory bandwidth: 177 Gbytes/sec. Double precision performance: 665 Gflops. Memory: 6 Gbytes.