Lecture 1

Introduction
Welcome to CSE 260!

• Your instructor is Scott Baden
  ◆ baden@ucsd.edu
  ◆ Office hours in EBU3B Room 3244
    ◆ Mondays 2.30 to 3.30pm; Thurs 4:30pm-5:30pm

• Your TA is Han Kim
  ◆ hskim@ucsd.edu
  ◆ Contact hours TBD

• The class home page is
  http://www.cse.ucsd.edu/classes/fa10/cse260

• CSME?

• Moodle

• Enrollment
Background Markers

- C/C++  Java  Fortran?
- Navier Stokes Equations
- Sparse factorization
- TLB misses
- Multithreading
- MPI
- CUDA, GPUs
- RPC
- Abstract base class

\[ \nabla \cdot u = 0 \]

\[ \frac{D\rho}{Dt} + \rho(\nabla \cdot v) = 0 \]

\[ f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \ldots \]
Text and readings

• Required texts

• Assigned class readings will include handouts and on-line material

• Texts on reserve in the S&E library

• Lecture slides

http://www.cse.ucsd.edu/classes/fa10/cse260/Lectures
Course Requirements

• Programming labs: 75%
  ◆ Teams of 2 or 3
  ◆ Includes a lab report, counts for half the grade
  ◆ Find a partner using the “looking for a partner” Moodle forum

• Class participation: 10%
  Be prepared to discuss the readings in class

• Midterm (week 7): 15%
Policies

• Academic Integrity
  ◆ Do you own work
  ◆ Plagiarism and cheating will not be tolerated

• By taking this course, you implicitly agree to abide by the following the course polices:
  www.cse.ucsd.edu/classes/fa10/cse260/Policies.html
Programming Laboratories

• 5 labs
  ♦ #1: Performance programming a single core
  ♦ #2 and 3: Nvidia Tesla (CSE and NCSA) – CUDA
  ♦ #4 MPI on a cluster (SDSC/NCSA) - MPI
• Lab #5: pick MPI or CUDA (or both)
• Teams of 2 or 3
• Teams of 3 will have a larger workload than teams of 2
• Establish a schedule with your partner from the start
Course overview and background

- How to solve computationally intensive problems on parallel computers
  - Software techniques
  - Performance tradeoffs

- Background
  - Graduate standing
  - Recommended: computer architecture (CSE 240A)
  - Students outside CSE are welcome
  - See me if you are unsure about your background

- Prior experience
  - Parallel computation?
  - Numerical analysis?
Syllabus

• Fundamentals
  Motivation, system organization, hardware execution models, limits to performance, program execution models, theoretical models

• Software and programming
   Programming models and techniques: message passing, multithreading
   Architectural considerations: GPUs and multicore
   Higher level run time models, language support
   CUDA, openmp, pthreads, MPI

• Parallel algorithm design and implementation
   Case studies to develop a repertoire of problem solving techniques: discretization, sorting, linear algebra, irregular problems
   Data structures and their efficient implementation: load balancing and performance
   Performance tradeoffs, evaluation, and tuning

© 2010 Scott B. Baden /CSE 260/ Fall '10
What is parallel processing?

• Decompose a workload onto *simultaneously executing physical resources*

• Multiple processors co-operate to process a related set of tasks – *tightly coupled*

• Improve some aspect of performance
  - Speedup: 100 processors run $\times 100$ faster than one
  - Capability: Tackle a larger problem, more accurately
  - Algorithmic, e.g. search
  - Locality: more cache memory and bandwidth

• Virtual or physical

• Reliability more of an issue at the high end or in critical applications
Granularity

• A measure of how often a computation communicates, and what scale
  ♦ Distributed computer: a whole program
  ♦ Multicomputer: function, a loop nest
  ♦ Multiprocessor: + memory reference
  ♦ Multicore: similar to a multiprocessor but perhaps finer grained
  ♦ GPU: kernel thread
  ♦ Instruction level parallelism: instruction, register
The impact of technology
Why is parallelism inevitable?

- Physical limits on processor clock speed and heat dissipation
- A parallel computer increases memory capacity and bandwidth as well as the computational rate

Average CPU clock speeds (via Bill Gropp)  http://www.pcpitstop.com/research/cpu.asp

© 2010 Scott B. Baden /CSE 260/ Fall '10
Today’s laptop would have been yesterday’s supercomputer

- Cray-1 Supercomputer
- 80 MHz processor
- 8 Megabytes memory
- Water cooled
- 1.8m H x 2.2m W
- 4 tons
- Over $10M in 1976

- MacBook
- 2.4GHz Intel Core 2 Duo
- 4 Gigabytes memory, 3 Megabytes shared cache
- NVIDIA GeForce 320m
  256MB shared DDR3 SDRAM
- Wireless Networking
- Air cooled
- ~ 2.7 x 33 x 23 cm. 2.1 kg
- $1149 in Sept. 2010
Technological disruption

• Transformational: modelling, healthcare…
• New capabilities
• Changes the common wisdom for solving a problem including the implementation

- Cray-1, 1976, 240 Megaflops
- Connection Machine CM-2, 1987
- Beowulf cluster, late 1990s
- Nvidia Tesla, 4.14 Tflops, 2009
- ASCI Red, 1997, 1Tflop
- Sony Playstation 3, 150 Gflops, 2006
- Tilera 100 core processor, 2009
- Intel 48 core processor, 2009

© 2010 Scott B. Baden /CSE 260/ Fall '10
Trends in Top 500 supercomputers
The age of the multi-core processor

• On chip parallel computer
• IBM Power4 (2001), many others follow (Intel, AMD)
• First dual core laptops (2005-6)
• GPUs (nVidia, ATI): a supercomputer on a desktop
The GPU

- Specialized many-core processor
- Many simple cores
- Reduced on-chip memory per core
- Explicitly manage the memory hierarchy
The impact

• *You* are taking this class
• A renaissance in parallel computation
• Parallelism is no longer restricted to the HPC cult with large machine rooms, it is relevant to everyone
• In a few years, everyone will have an historically unprecedented amount of parallelism at their disposal
• May or may not need to know they are using a parallel computer
Motivating Applications
Simulates a 7.7 earthquake along the southern San Andreas fault close to LA using seismic, geophysical, and other data from the Southern California Earthquake Center

**How it works:**

1. **Divide up Southern California into “blocks”**
2. **For each block, get all the data on ground surface composition, geological structures, fault information, etc.**
How TeraShake Works

3. Map the blocks on to processors of the supercomputer
4. Run the simulation using current information on fault activity and the physics of earthquakes
Application #2: modeling the heart on a GPU

• Fred Lionetti and Andrew McCulloch
• nVidia GTX-295, single GPU: 240 single precision units @ 1.242 GHz
• \( \times 134 \) speedup compared with openmp running on quad core i7 @ 2.93GHz
• \( \times 20 \) improvement over entire simulation
Application #3: Face detection on a GPU

- Jason Oberg, Daniel Hefenbrock, Tan Nguyen [CSE 260, fa’09 →fccm’10]
- Meets or exceeds performance of FGPA on VGA images

Features

Sonia Sotomayor
How do we know if we’ve succeeded?

• Capability
  ♦ Solve a problem under conditions that were not possible previously

• Performance
  ♦ Solve the same problem in less time than before
  ♦ This can provide a capability if we are solving many problem instances

• The result achieved must justify the effort
  ♦ Enable new scientific discovery
Is it that simple?

• Simplified processor design, but more user control over the hardware resources
  - Redesign the software
  - Rethink the problem solving technique

• If we don’t use the parallelism, we lose it
  - Amdahl’s law - serial sections
  - Von Neumann bottleneck
  - Load imbalances
Why is parallel programming challenging?

• A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be reformulated numerically

• There is no magic compiler that can turn a serial program into an efficient parallel program all the time and on all machines
  - Performance programming involving low-level details: heavily application dependent
  - Irregularity in the computation and its data structures forces us to think even harder
  - Users don’t start from scratch-they reuse old code. Poorly structured code, or code structured for older architectures can entail costly reprogramming
The two mantras for high performance

• Domain-specific knowledge is important in optimizing performance, especially locality

• Significant time investment for each 10-fold increase in a performance
Performance and Implementation Issues

• Data motion cost growing relative to computation
  ‣ Conserve locality
  ‣ Hide latency
• Little’s Law [1961]

\[ \text{# threads} = \text{performance} \times \text{latency} \]

\[ T = p \times \lambda \]

\[ p \text{ and } \lambda \text{ increasing with time} \]
\[ p = 1 \text{ - 8 flops/cycle} \]
\[ \lambda = 500 \text{ cycles/word} \]