Lecture 1

Introduction
Welcome to CSE 260!

• Your instructor is Scott Baden
  - baden@ucsd.edu
  - Office hours in EBU3B Room 3244
    - Tues/Thurs 4pm to 5pm or by appointment

• Your TA is Jing Zheng
  - j3zheng@ucsd.edu

• The class home page is
  - http://www.cse.ucsd.edu/classes/wi14/cse260-a

• CSME?

• Logins – www-cse.ucsd.edu/classes/wi14/cse260-a/lab.html
  - Moodle
  - Bang
  - Dirac
  - Lilliput
Text and readings

• Required texts
  ♦ *An Introduction to Parallel Programming*, by Peter Pacheco, Morgan Kaufmann (2011)

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

• Lecture slides

  [www.cse.ucsd.edu/classes/wi14/cse260-a/Lectures](http://www.cse.ucsd.edu/classes/wi14/cse260-a/Lectures)
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/wi14/cse260-a/Policies.html
Course Requirements

• Do the readings **before** lecture and be prepared to discuss in class

• 3 Programming labs
  ♦ Teams of 2 or 3, Teams of 1 with permission
  ♦ May switch teams but be sure to give notice as explained in *Course Policies*
  ♦ Includes a lab report, greater emphasis (grading) with each lab
  ♦ Find a partner using the “looking for a partner” Moodle forum
Programming Laboratories

• 3 labs
  - A1: Performance programming a single core
  - A2: GPU Fermi (NERSC’s Dirac) – CUDA
  - A3: MPI on a cluster (Bang) - with options

• A3 is worth 50%, one month duration
  - More discussed later in the quarter
• Teams of 2 or 3
• Establish a schedule with your partner at the start
Course overview and background

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

• Background
  ◆ Graduate standing
  ◆ C or C++
  ◆ 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?
Background Markers

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

\[ \nabla \cdot u = 0 \]
\[ \frac{D \rho}{Dt} + \rho (\nabla \cdot v) = 0 \]
\[ f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... \]
What you’ll learn in this class

• How to solve computationally intensive problems on parallel computers effectively
  - Theory and practice
  - Software techniques
  - Performance tradeoffs

• Different platforms: GPUs, cluster

• CSE 260 will build on what you learned earlier in your career about programming, algorithm design & analysis and generalize them
Syllabus

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

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

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

• Learning is not a passive process
• I won’t just “broadcast” the material
• Consider the slides as talking points, class discussions driven by your interest
• Class participation is important to keep the lecture active
• Complete the assigned readings before class
• Different lecture modalities
  ♦ The 2 minute pause
  ♦ In class problem solving
The 2 minute pause

• Opportunity **in class** to develop understanding, make sure you “grok” it
  ♦ By trying to explain to someone else
  ♦ Getting your brain actively working on it

• What will happen
  ♦ I pose a question
  ♦ You discuss with 1-2 people around you
    • Most important is your understanding of why the answer is correct
  ♦ After most people seem to be done
    • I’ll ask for quiet
    • A few will share what their group talked about
      – Good answers are those where you were wrong, then realized…
The rest of the lecture

• Syllabus
• Intro to parallel computation
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
- Resilience an issue at the high end
  - Exascale: $10^{18}$ flops/sec
Parallel Processing, Concurrency & Distributed Computing

- **Parallel processing**
  - Performance (and capacity) is the main goal
  - More tightly coupled than distributed computation

- **Concurrency**
  - Concurrency control: serialize certain computations to ensure correctness, e.g. database transactions
  - Performance need not be the main goal

- **Distributed computation**
  - Geographically distributed
  - Multiple resources computing & communicating unreliably
  - “Cloud” computing, large amounts of storage
  - Looser, coarser grained communication and synchronization

- May or may not involve separate physical resources, e.g. multitasking “Virtual Parallelism”
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
Today’s mobile computer would have been yesterday’s supercomputer

- Cray-1 Supercomputer
- 80 MHz processor
- 240 Mflops/sec peak
- 3.4 Mflops Linpack
- 8 Megabytes memory
- Water cooled
- 1.8m H x 2.2m W
- 4 tons
- Over $10M in 1976

- iPad 4
- Apple A6X SoC
  - 1.4GHz dual-core Apple swift processor (4600 Megaflops)
  - Quad-core PowerVR SGX 554 GPU
- 988 Megabytes of memory, 32KB L1 and 1MB L2 caches
- 64GB Flash storage
- Color display
- Wireless or phone Networking
- Air cooled
- ~ 9.4 x 186 x 241 mm, 652 g
- $699 in September 2013

©2014 Scott B. Baden / CSE 260 / Winter '14
The progress of 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
- Intel Phi, 50 cores, 2012
- Nvidia Tesla, 4.14 Tflops, 2009
- ASCI Red, 1997, 1Tflop
- Sony Playstation 3, 150 Gflops, 2006
- Apple A6X, 2012
The age of the multi-core processor

- On-chip parallel computer
- IBM Power4 (2001), many others follow (Intel, AMD, Tilera, Cell Broadband Engine)
- First dual core laptops (2005-6)
- GPUs (nVidia, ATI): desktop supercomputer
- In smart phones, behind the dashboard blog.laptopmag.com/nvidia-tegrak1-unveiled
- Everyone has a parallel computer at their fingertips
- If we don’t use parallelism, we lose it!
The GPU

- Specialized many-core processor
- Massively multithreaded, long vectors
- 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
• May or may not know when using a parallel computer
Have you written a parallel program?

- Threads
- MPI
- RPC
- C++11 Async
- CUDA
Motivating Applications
A Motivating Application - TeraShake

Simulates a 7.7 earthquake along the southern San Andreas fault near LA using seismic, geophysical, and other data from the Southern California Earthquake Center

epicenter.usc.edu/cmeportal/TeraShake.html
How TeraShake Works

- Divide up Southern California into blocks
- For each block, get all the data about geological structures, fault information, …
- Map the blocks onto processors of the supercomputer
- Run the simulation using current information on fault activity and on the physics of earthquakes
Animation
The Payoff

• Capability
  ◆ We solved a problem that we couldn’t solve before, or 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
  ◆ Software costs must be reasonable
I increased performance – so what’s the catch?

• 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
What’s involved

• Performance programming
  ◆ Low-level details: heavily application dependent
  ◆ Irregularity in the computation and its data structures forces us to think even harder

• Simplified processor design, but more user control over the hardware resources

• Parallelism introduces many new tradeoffs
  ◆ Redesign the software
  ◆ Rethink the problem solving technique
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 performance
Performance and Implementation Issues

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

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

\[ T = p \times \lambda \]

- \( p \) and \( \lambda \) increasing with time
  - \( p = 1 - 8 \) flops/cycle
  - \( \lambda = 500 \) cycles/word