Lecture 1

Introduction to parallel computing
Welcome to CSE 160!

• Your instructor is Scott B. Baden  
  baden@cs.ucsd.edu

• Office: room 3244 in EBU3B

• Office hours
  – Week 1: Mon 11-12, Thu 4-5, or by appointment
  – Remaining weeks: TBD

• Your TA is Pietro Cicotti
  – Lab and office hrs TBA (No section hours this week)

• Section
  – Friday 1:00 to 2:00 pm, CENTR 216
  – No section this week
Content

• Our home page is  
  http://www.cse.ucsd.edu/classes/fa08/cse160

• All class announcements will be made on-line so check this web page frequently

• Web board - contact ACS if you cannot log in

• One required text:  
  Parallel Programming, by Calvin Lin and Larry Snyder.  
  Addison-Wesley, ISBN 0321487907

• Useful information on-line  
  http://www-cse.ucsd.edu/users/baden/Doc/
Background

• Required pre-requisite
  – CSE 100 or Math 176

• C/C++ or Fortran programming experience

• Past experience in parallel computation?

• Threads programming? MPI? RPC? Sockets?

• Numerical analysis?
Course Requirements

• 5 Homework assignments (50%)
  – Includes programming labs and “pencil and paper” work, e.g. problem sets
  – Programming shall be done in teams

• Exams (35%)
  – Midterm (15%)
  – Final (20%)

• Weekly quizzes in section (15%)
Policies

• By taking this course, you implicitly agree to abide by the following the course polices: http://www.cse.ucsd.edu/classes/fa08/cse160/Policies.html

• Academic Honesty
  – Do you own work
  – Plagiarism and cheating will not be tolerated

• See other policies on-line
Scheduling

• Some lectures will be rescheduled
  – Thursday 10/9 → Friday 10/10      Time TBD
    Swap with section
  – Friday 10/17: make up lecture (11/17)  TBD
  – Thursday 11/20 → Friday 11/21      Time TBD
    Swap with section
Hardware platform

• Multi-core processors
• For your first programming assignment you may use any machine that runs pthreads, including your personal laptop
• This will be discussed in section
Course overview

• Theory and practice of parallel computation
• Emphasis on multi-core computation
• Programming with threads
• Case studies to develop a repertoire of problem solving techniques
What is parallel computation and why is it inevitable?

• Simultaneous computation over separate physical resources to increase capacity or speed
• Multiple processors co-operate to solve a related set of tasks comprising a “single” problem
• Reliable, tightly coupled interactions
• Two view points
  – Speedup: 100 processors run \( \times 100 \) faster than one
  – Opportunity: Tackle a larger problem, more accurately
• A parallel computer increases memory capacity and bandwidth as well as the computational rate
Why is parallel programming interesting?

• 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
How does parallel computing relate to other branches of computer science?

- Parallel processing generalizes problems we’ve encountered on single processor computers
- A parallel computer is just an extension of the traditional memory hierarchy
- The need to preserve locality, which prevails in virtual memory, cache memory, and registers, also applies to a parallel computer
A Motivating Application
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.

Slide Courtesy of DataCentral@SDSC
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
Resources must support a complicated orchestration of computation and data movement

240 procs on SDSC Datastar, 5 days, 1 TB of main memory

Continuous I/O 2GB/sec

47 TB output data for 1.8 billion grid points

Parallel file system

Data parking

Data parking of 100s of TBs for many months

Large memory Nodes with 256 GB of DS for pre-processing and post visualization

10-20 TB data archived a day

The next generation simulation will require even more resources: Researchers plan to double the temporal/spatial resolution of TeraShake

"I have desired to see a large earthquake simulation for over a decade. This dream has been accomplished."

Bernard Minster, Scripps Institute of Oceanography

Slide Courtesy of DataCentral@SDSC
How do we know if we’ve succeeded?

• Opportunity
  – 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
Parallelism, concurrency, and distributed computing

• Concurrency
  – May or may not involve separate physical resources, e.g. multitasking “Virtual Parallelism”
  – Concurrency control: serialize certain computations to ensure correctness, e.g. database transactions
  – Performance need not be the main goal

• Distributed computation
  – Multiple resources communicating over unreliable media
  – Multiple launch sites, separate free standing programs
  – “Cloud” or “Grid” computing, large amounts of storage

• Granularity: how often a computation communicates, and what scale

• Distributed computation employs a coarser granularity than parallel computation
Blurring the distinction

• We may trivially execute a distributed computation on a parallel computer
  – Consider client-server, or work farms
  – Cloud or grid computing
  – This may not be the most effective way to use a parallel computer

• Multiple parallel computations may be coupled in a distributed fashion
Trends in High Performance and Parallel Computation

• Historically a small community
  – Scientific (HPC) users at national laboratories, later at NSF centers
  – Started in the 1970s: ILIAC IV, Cray-1, CM*, C.MMP
  – Impact in the early 1990s - higher fidelity simulations
  – Beowulf clusters in small laboratories

• Outside the HPC community
  – Web servers
  – Data bases
  – Process control

• Everyone benefits from steadily increasing CPU clock speeds, a consequence of Moore’s law (doubling of transistors on chip)
Today’s laptop would have been yesterday’s supercomputer

- Cray-1 Supercomputer
- 80 MHz processor
- 8 Megabytes memory
- Water cooled
- 6 feet H x 7 feet W
- 4 tons
- Over $10M in 1976

- MacBook
- 2.4GHz Intel Core 2 Duo
- 2 GBytes DDR2 SDRAM
- 4 Megabyte shared cache
- Air cooled
- 1.1 × 12.8 × 8.9 inches
- 5.0 pounds (2.3 kg)
- $1299 in 2008
We have reached a limit!

Physical limitations on processor clock speeds

Average CPU clock speeds (via Bill Gropp)  http://www.pcpitstop.com/research/cpu.asp
The age of the multi-core processor

- Achieve the effect of increased clock speeds by replicating the CPUs
  - IBM Power4 (2001)
  - Many others follow: Intel, AMD, etc.
  - GPUs: NVIDIA, ATI
  - Heterogeneous processors: Cell Broadband Engine (PS3)
  - Tilera

- Simplified processor design, more user control over the hardware resources

- Is it that simple?
  - Von Neumann bottleneck
  - Amdahl’s law bottleneck
  - Software issues
  - Performance tuning
The processor-memory gap