Lecture 1

Introduction to parallel computing

People and places

- Your instructor is Scott B. Baden
  - Office hours in APM 4141: Weds 2:00 to 4:00
- Your TA is Urvashi Rao
  - Office hrs M 1-2, Th 2-3 (No office hours this week)
- Section
  - Fridays 3:00 to 3:50 in CSB 004
- The class home page is http://www.cse.ucsd.edu/classes/sp04/cse160
- All class announcements will be made on-line so check this web page frequently
Background

• Required pre-requisite
  – CSE 100 or Math 176
• C/C++ or Fortran programming experience
• Past experience in parallel computation?
• Numerical analysis?

Texts and readings

• One required text:
  **Be sure to get the 1st edition, not the 2nd**

• One recommended text:
Course Requirements

- 5 or 6 Homework assignments (50%)
  - Includes programming and “pencil and paper” work, e.g. problem sets
  - All work will be done individually
- In class exams (35%)
  - Midterm (15%)
  - Final (20%)
- 3 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/sp04/cse160/Policies.html
- Academic Honesty
  - Do you own work
  - Plagiarism and cheating will not be tolerated
- See other policies on-line
Hardware platform

- Beowulf cluster: valkyrie.ucsd.edu
  - 16 nodes, each a dual 1Ghz Pentium III CPU
  - Each node has 1GB of RAM and runs Linux
  - A Myrinet switch provides low latency connectivity between the nodes.

- Complete a computer account form today if you don’t already have an account & login!

For more information

- Web pages for more documentation
  
  http://www.cse.ucsd.edu/classes/sp04/cse160/testbeds.html
  
  http://www- cse.ucsd.edu/users/baden/Doc/
Course overview

- Theory and practice of parallel computation
  - Parallel system design
  - Parallel algorithm design and implementation
  - Programming models and techniques: primarily message passing but some multithreading
  - Performance tradeoffs, evaluation, and tuning
  - Case studies to develop a repertoire of problem solving techniques

Today’s readings

- Text
  - Chapter 1: §1.1, 1.2.1, 1.3 (stop at Networks for Multicomputers), 1.4.1 (stop at Ethernet Connections)

- On line lecture notes at
  http://www.cse.ucsd.edu/classes/sp03/cse160/Lectures/Lec01.html
Demand for parallel computation

• A single processor computer does not meet the needs of computationally intensive problems
  – Insufficient memory and memory bandwidth
  – Insufficient computational capacity
• A parallel computer is…
  – a collection of concurrently executing, communicating processing elements
• Potential for a large performance boost: 100 processors run 100 times faster than one processor
• Or, tackle a problem $100 \times$ the size, accuracy…

A motivating application

• Oceanographic simulation
• Track pockets of water that “overturn” fluid that is less dense

Bill Smyth, Dept. Oceanic & Atmospheric Sciences, Oregon State University
Visualizing the overturn regions


The computation

- Compute on a cubical volume of water
- Split the volume into a mesh of points
- A $128^3$ problem runs in 10 hours on a single processor workstation, 256 MB memory
How big can the problems get?

- The *resolution* of the mesh limits our ability to examine small features
- We need higher resolution: $256^3$, $512^3$, $1024^3$
- When we double resolution, we increase memory requirements by $\times 8$, time by $\times 16$
  - A $256^3$ run would take 160 hr (1 wk) & 2 GB memory
  - A $1024^3$ run would take ~5 years & 128 GB of memory
- We need to make several runs (5 to 10)
- We cannot solve larger problems on a single processor implemented with today’s technology

Hardware Technology

- Processor speeds double in ~18 months
- Improving at a *much* faster rate than DRAM access times
  - 55% per year vs. 7% per year
  - Memory bandwidth is decreasing relative to processing speed and this has lead to a phenomenon called the “memory wall”
  - Interprocessor communication speed is limited by memory bandwidth
The processor-memory gap

The advance of technology

• Today’s PC would have been the most powerful supercomputer in 1976
  – Cray-1: 80 MHz clock, 10s of megabytes of memory
  – Laptop in 2003: 4+GHz clock, hyperthreading, 1 GB of memory, 60GB of disk
• A scientist could simply wait for a faster processor to come along rather than enhance their program
  – In 2008 we might be able to run our $256^3$ problem in 10 hours on a single CPU
Opportunity

- Not practical to wait for CPU clock speed increase to the desired performance level
  - We’d need to wait until 2020 to realize $512^3$ resolution
  - Won’t solve the memory bandwidth problem
- Idea: connect many processors together to harness their collective capacity including memory bandwidth, cache size, as well as processing
- Makes sense when the problems are truly large
- Large can mean space and/or time

A generic parallel computer

- A collection of $P$ processor-memory pairs
- Processors communicate over the interconnect
- We might not know what the interconnect looks like
Parallel processing

• We define parallel processing as the simultaneous execution or overlap of independent processing activities
• Different levels of granularity
• Granularity dictates how often a computation communicates, and what scale
  – Distributed computer: at the level of a whole program
  – Multicomputer: function, a loop nest
  – Multiprocessor: + memory reference (cache line)
  – Instruction level parallelism: instruction, register

Relationship to distributed and “grid” computing

• Parallel processing covers a wide range of activities and includes some aspects of distributed and grid computing
• With parallel processing there is a greater degree of coupling (interaction) between processors and memory compared with distributed computing
• Our parallel computations generally
  – communicate more frequently and for shorter durations than distributed computations (finer grained)
  – don’t tolerate significant external contention from other jobs’ use of processors, memory, and interconnect
  – don’t need to form a consensus
  – don’t address security issues beyond what would be encountered in a serial implementation
  – don’t address failure beyond what that of a serial computation (the failure of even one processor will cause the entire computation to fail)
Address Space Organization

- Our generic machine model includes an option for global memory
- We classify the address space organization of a parallel computer according to whether or not it provides global memory
- When there is a global memory we have a “shared memory” architecture, also known as a *multiprocessor*
- Where there is no global memory, we have a “shared nothing” architecture, also known as a *multicomputer*

Shared memory organization

- The address space is global to all processors
- Hardware automatically performs the global to local mapping using address translation mechanisms
- We classify shared memory architectures still further according to the uniformity of memory access times
- Costly to build larger systems with hundreds of processors
Two kinds of shared memory

• We classify shared memory architectures still further according to the uniformity of memory access times

• UMA: Uniform Memory Access time
  – All processors observe the same access time to memory in the absence of contention (approximates a PRAM)
  – These are also called *Symmetric Multiprocessors*
  – Usually bus based: not a scalable solution

NUMA

• Non-Uniform Memory Access time
  – Processors see distant-dependent access times to memory
  – Implies physically distributed memory

• We often call these *distributed shared memory architectures*
  – Commercial example: SGI Origin 3000, up to 512 processors
  – Elaborate interconnect with a directory structure to monitor sharers
But why do we use shared memory?

- A natural extension for existing single processor execution model
- We don’t have to distinguish local from remote data
- More natural for the programmer 😊
- But an efficient program, may end up having to mimic a message passing program!

Cost of supporting shared memory

- For a given level of aggregate memory bandwidth, shared memory appears more costly than architectures that do not support the capability
Architectures without shared memory

- Some architectures have no global memory
- Each processor has direct access to local memory only
- Send and receive messages to obtain copies of data from other processors
- We call this a *shared nothing* architecture, or a *multicomputer*

Message Passing

- Message based communication requires that the sender and receiver be aware of one another
- There must be an explicit recipient of the message
- In message passing there are two events:
Hybrid organizations

- Multi-tier organizations are hierarchically organized
- Each node is a multiprocessor
- Nodes communicate by passing messages, processors within a node communicate via shared memory
- IBM SP systems at SDSC and NERSC

Running our computation in parallel

- Divide the ocean and the underlying mesh into sub-problems
- Solve problem in each sub-problem locally
- Communicate information between the sub-problems
**Issues**

- Communication and coordination are not free
- The best way to handle them depends on the hardware
- Each processor must receive a fair share of the work, or the benefit of using multiple processors will be limited to that of the heavily loaded processor(s)
- Some parts of our task can’t run on multiple processors, or may not parallelize as well as others: these limit performance asymptotically

**How do we know if we’ve succeeded?**

- Capability
  - Solve a problem under conditions that were previously not possible
- 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
Justifying the effort

- Programmer time and user time are often more valuable than machine time
- We should make the effort to improve performance or memory consumption
  - If we expect to run the application many times
  - If doing so enables us to improve some aspect of solving the problem (accuracy, precision, etc.)
- The time spent improving some aspect of program behavior must be commensurate with the benefits

Why can parallel programming be difficult?

- 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
  - We need to manage low-level activities like interprocessor communication
  - 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 user’s perspective

- Users are more concerned about solving the problem than improving performance
- They may be more concerned with managing the large amounts of data their application produces
- Employ the minimal amount of parallelism needed to get the job done
  - If a problem runs overnight, then the user may not notice a distinction between an 8 hour vs a 16 hour run when they halve the number of processors, other than…
  - That 8 hour run might actually run in 12 – 14 hours rather than 16. Why?
  - Memory is more of a constraint than processing speed

An important universal: the locality principle

- Programs generally exhibit two forms of locality when accessing memory
  - Temporal locality (time)
  - Spatial locality (space)
- In scientific problems, locality often involves loops
- These principles apply to optimizations that improve re-use in cache, but they also apply to managing re-use with local and non-local memory
Technology

• Large memories are slow and cheap
  – Disk
  – DRAM (Dynamic random access memory)
  – Thousands of times faster than disk
• Small memories are fast but expensive
  – SRAM (Static random access memory)
  – 5 or 10+ times slower than DRAMs
Why do we have memory hierarchies?

- Processor speeds are improving at a much faster rate than DRAM access times
- 55% per year v. 7% per year
- We have a severe mismatch at the processor memory interface
- Faster (and more expensive) memories are smaller than slower (and less costly ones)
- We may use a hierarchy to effectively reduce the cost of accessing memory: cache, TLB, etc.

Idea

- Put recently accessed data in a small, fast memory
- If this memory is 10 times faster (access time) than main memory …
- And if we find what we are looking for 90% of the time…
- Access time = 0.90 × 1 + (1-0.9) × 10
  = 1.9
- Memory is now 5 times faster
Sidebar

- If cache memory access time is 10 times faster than main memory …
- \( T_{\text{cache}} = \frac{T_{\text{main}}}{10} \)
- And if we find what we are looking for \( f \times 100\% \) of the time…
- Access time = \( f \times T_{\text{cache}} + (1- f) \times T_{\text{main}} \)
  = \( f \times \frac{T_{\text{main}}}{10} + (1- f) \times T_{\text{main}} \)
  = \( (1-(9f/10)) \times T_{\text{main}} \)
- We are now \( 1/(1-(9f/10)) \) times faster
- To simplify, we use \( T_{\text{cache}} = 1, T_{\text{main}} = 10 \)

Some terminology

- If we find what we are looking for, we have a **cache hit**
- If we don’t find what we are looking for, we have a **cache miss**
- The time it takes to access cache on a hit is the **hit time**
- The time it takes to obtain the data we need from the next higher level of the hierarchy is the **miss penalty**
More terminology

• The percentage of time that we hit in the cache is called the hit ratio
• The miss ratio is (1 – hit ratio)
• In today’s architectures we have on-chip cache(s) and external cache(s)
• Internal cache often contains a separate data cache and instruction cache
• External cache is often unified
• The unit of transfer is called the block

Valkyrie’s memory hierarchy

• There are two levels of cache: L1 and L2
• If what we are looking for isn’t in L1…
• We look in L2
• If what we are looking for isn’t in L2..
• We look to main memory
• Which is larger?
  – Size of L1: 32 KB
  – Size of L2: 256 KB
How does parallel computing relate to other branches of computer science?

- A parallel computer is just an extension to the traditional memory hierarchy
- The notions of locality, which prevail in virtual memory, cache memory, and registers, also apply
- We seek to preserve locality inherent to the application, and to amortize fixed costs

Memory Hierarchies

- We can think of the collective memory of all the processors in a parallel computer as another level of the memory hierarchy: Alpert et al. “Modeling Parallel Computers as Memory Hierarchies,” Programming Models for Massively Parallel Computers, 1993