Parallel Computing I

4003-531/4005-735 Section 01

TR 4:00pm – 5:50pm
GOL(70)-2690

Prof. Matthew Fluet

Parallel Computing I

Introduction and Overview

Course Description

Parallel Computing is the study of the hardware and software issues in parallel computing. Topics include an introduction to the basic concepts, parallel architectures and network topologies, parallel algorithms, parallel metrics, parallel languages, granularity, applications, parallel programming design and debugging. Students will become familiar with various types of parallel architectures and programming environments. Programming projects will be required.

Course Administration

Instructor: Matthew Fluet

- E-mail: `mtf@cs.rit.edu`
- Office: GOL(70)-3555
- Office hours: W 3pm–5pm, R 9am–11am, F 1pm–3pm

Website

- `http://www.cs.rit.edu/~mtf/teaching/20102/pc1`
- `http://mycourses.rit.edu`

Introductions

Who am I?

- Matthew Fluet
- 2nd year faculty member (Dept. of CS)
- Research interests lie with programming languages, including compiler technology, parallelism and concurrency, type systems, and program semantics.
  - Cyclone (a safe dialect of C)
    - Safe(er) parallel programming (but not my work)
  - MLton (a Standard ML compiler)
    - Consulted on various extensions to support parallelism
    - Interested in folding one of those research prototypes into the production system
  - Manticore (a heterogeneous parallel functional language)

Assignments, Exams, & Grades

- 10% — Attendance & Participation
- 25% — Homeworks/Programming Assignments (≈ 4)
- 25% — Term Team Project
- 20% — Mid-term exam (Jan. 11 (Tue), in class)
- 20% — Final exam (Feb. 24 (Thur) @ 12:30pm)

More details on syllabus

Attendance & Participation

Attendance is strongly encouraged

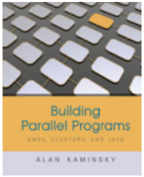
- Will post slides to website
  - (but trying to use slides less and board more)
- Take notes
  - (not everything from a lecture is in the slides and/or texts)

Participation means being an engaged student

- Asking and answering questions
  - (not simply attending class)
- Let me know if pace is too fast or too slow
- When I enter your grade, will I know who you are?

Textbook and Software

Building Parallel Programs: SMPs, Clusters, and Java, Alan Kaminsky



Parallel Java Library, Alan Kaminsky

- an API and middleware for parallel programming in 100% Java on shared memory multiprocessor (SMP) parallel computers, cluster parallel computers, and hybrid SMP cluster parallel computers.

Term Team Project

Students will underake a term project in teams of 2 or 3 students. The term project will consist of proposing a topic for investigation, developing a parallel program and measuring its performance characteristics, submitting the developed software and a written report, and giving a presentation and demo during class. Students taking the course for graduate credit will be expected to include a more significant research component than those taking the course for undergraduate credit.

December 6 (Mon.):	Project team formation due
December 13 (Mon.):	Project proposal due
February 14 (Mon.):	Project software and report due
February 15&17 (Tue.&Thu.):	Project presentations
February 17 (Thu.):	Project presentation materials due

Parallel Computing: Bigger Problems & Faster Answers

Many computer applications require massive amounts of computation:

- Weather forecasting
- Climate modeling
- Protein sequence matching
- Star cluster simulation
- Game AI (Chess, Go)
- Animation and special effects

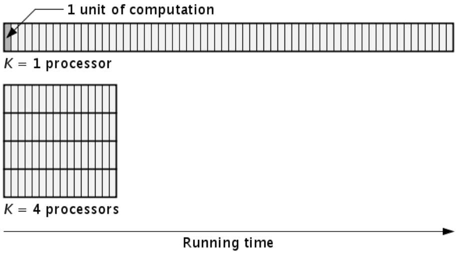
Utility of these applications increases as they are able

- to handle bigger problems, and
- to produce faster answers.

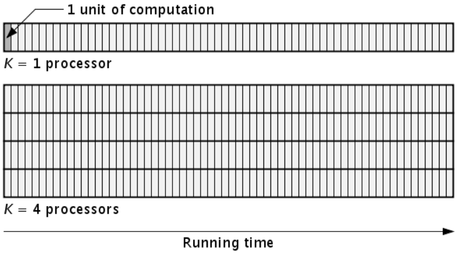
One option: Multiple computers work on the computation simultaneously.

Parallel Computing: Bigger Problems & Faster Answers

- Faster answers (speedup)



- Bigger problems (sizeup)



Digression: Parallelism vs. Concurrency

One take on this (terminology not universal, but distinction important):

*Software is **parallel** if a primary intellectual challenge is using extra computational resources to do more useful work per unit time.*

- Examples: scientific computing, most graphics, a lot of servers
- Key challenge is Amdahl's Law (no sequential bottlenecks)
- Often provide parallelism via threads on different processors
- Ideally deterministic, but not when concurrent

*Software is **concurrent** if a primary intellectual challenge is responding to external events from multiple sources in a timely manner.*

- Examples: operating system, GUI, version control
- Key challenge is responsiveness (interactivity)
- Often provide responsiveness via threads
- Inherently non-deterministic, but not necessarily parallel

Another Digression: Huge Topic

Parallel computing is a *huge* topic.

Lots of different directions one could go.

Lots of recent changes in hardware and software.

We're scratching the surface and laying a foundation.
Make you better able to pursue these topics on your own.

A Brief History of Parallel Computing

Hardware

- Then (< 1995): Mainly specialized hardware
  - Specialized processors – vector processors, Transputers, Connection Machine, etc.
  - Specialized interconnections – mesh, torus, hypercube, omega, butterfly, etc.
- Four (?Five?) significant events
  - Ethernet local area network (late 1970s)
  - TCP/IP protocoal (1981)
  - IBM PC open architecture (1981), becomes de facto standard
  - Linux operating system (1991)
  - (? Intel dual-core Pentium (2005) ?)
- Now: Mainly commodity hardware
  - Commodity processors – single- or multiple-processor PCs
  - Commodity interconnections – Ethernet switches
  - Beowulf (1995), Stone SouperComputer (1996)

A Brief History of Parallel Computing

Supercomputers

- 1993: only 104 of the top 500 supercomputers used commodity CPU chips (like Intel x86 or Sun SPARC).
- 2008: 498 of the top 500 supercomputers used commodity CPU chips (AMD, IBM and Intel).
- November 2010: 3 of the top 10 supercomputers include NVidia GPUs

A Brief History of Parallel Computing

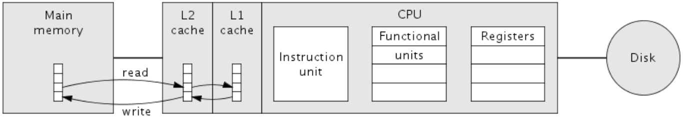
Software

- Then: At the language level, no standards
  - Parallelizing compilers for sequential languages
  - New parallel languages – e.g. occam, Linda
  - Parallel extensions to sequential languages – e.g. Fortran 90
- Now: At the middleware level (above the language level), standards-based
  - OpenMP (SMP)
  - MPI (Cluster)
  - Parallel Java (SMP and Cluster)

Review: CPU and Computer Architecture

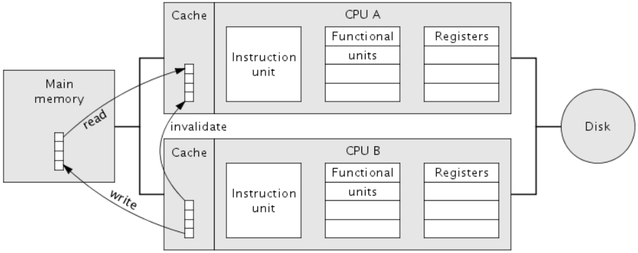
Many ways to increase performance of single processor computer systems:

- CPU
  - Increase clock speed
  - Pipelined architecture
  - Superscalar architecture
  - Instruction reordering
- Memory
  - Decrease access times
  - Caches

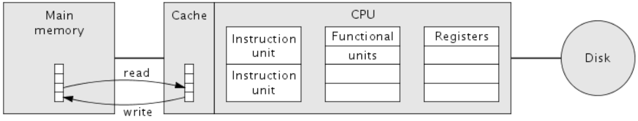


Review: Commodity Parallel CPUs

- Symmetric Multiprocessor (SMP)



- HyperThreading



- MultiCore
  - Like SMP, but sharing memory bus and some levels of cache

Parallel Computers

Main classes:

- Symmetric Multiprocessor (SMP) Parallel Computers
- Cluster Parallel Computers
- Hybrid Parallel Computers

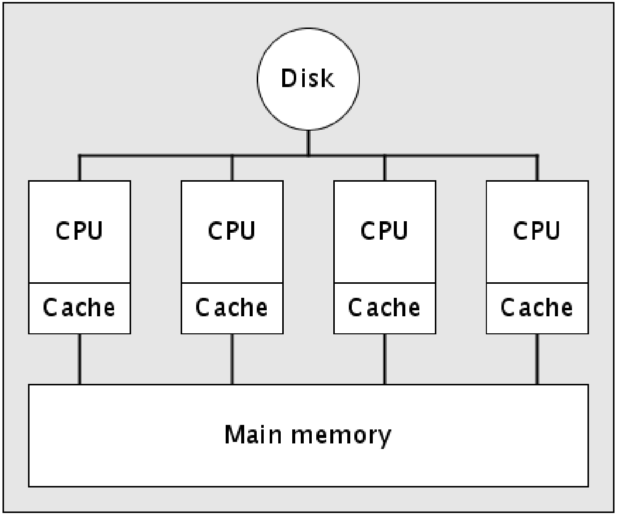
Main issues:

- Architecture
- Communication

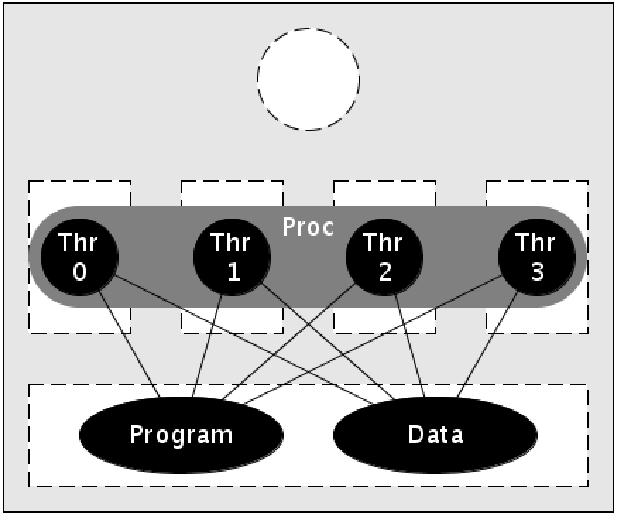
Other classes:

- Grid Parallel Computers
- General-Purpose GPU (GPGPU) Parallel Computers

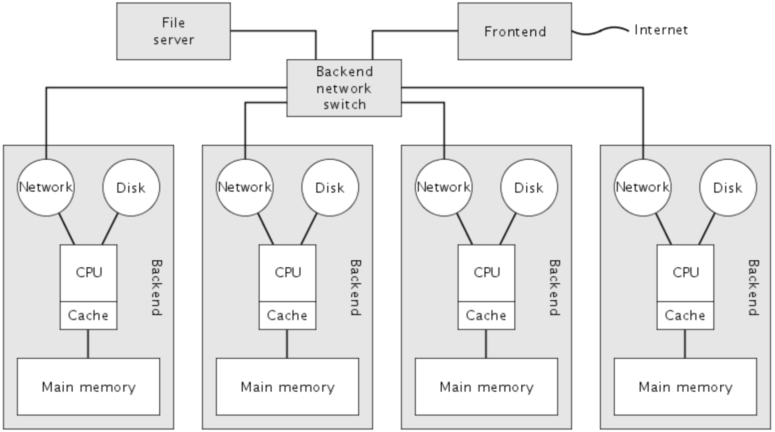
SMP Parallel Computers: Architecture



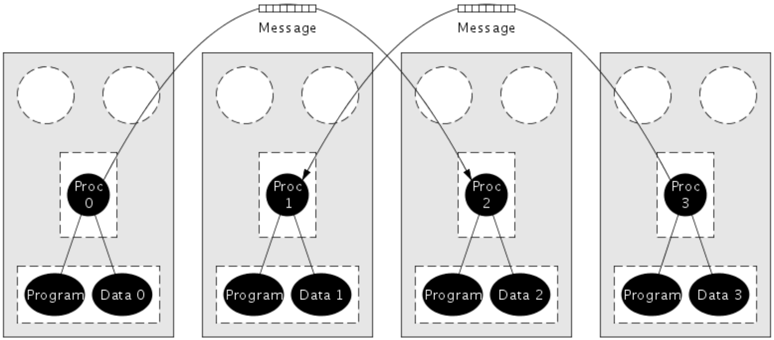
SMP Parallel Computers: Communication



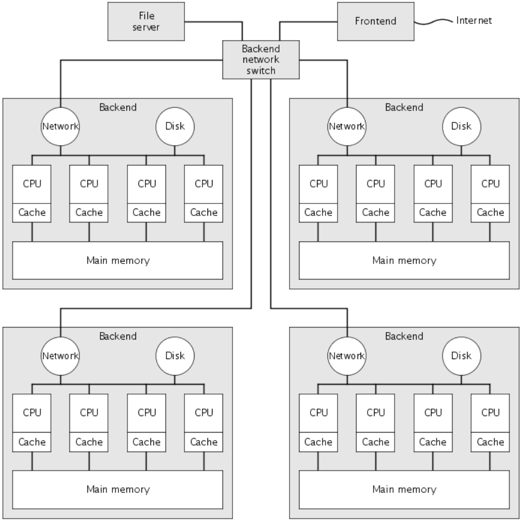
Cluster Parallel Computers: Architecture



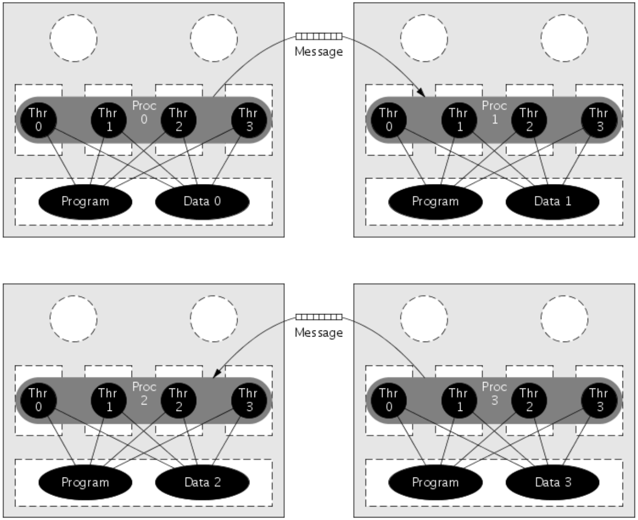
Cluster Parallel Computers: Communication



Hybrid Parallel Computers: Architecture



Hybrid Parallel Computers: Communication



SMP vs Cluster Parallel Computers

SMP vs Cluster Parallel Computers

- SMP Parallel Computers
  - Pro: Better suited for problems with data dependencies between processors
  - Con: Limited memory and CPU scalability
  - Con: Worse price/performance ratio
- Cluster Parallel Computers
  - Pro: No limits on scalability of memory and CPU
  - Pro: Better price/performance ratio
  - Con: Messaging overhead
- Hybrid Parallel Computers
  - Best of both worlds?

My Office Machines

MacPro

- One Intel Xeon quad-core w/ 2-way hyperthreading (8 processors), 2.66 GHz clock, 6 GB main memory

MacBook Pro

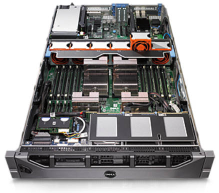
- One Intel Core 2 Duo (2 processors), 2.53 GHz clock, 3 GB main memory

RIT CS Parallel Computers

- SMPs
  - `paradise`, `paradox`, `paragon`, `parasite`
    - Four Sun UltraSPARC-IV dual-core CPUs (eight processors), 1.35 GHz clock, 16 GB main memory
- Cluster
  - `paranoia`: frontend computer
    - UltraSPARC-II CPU, 296 MHz clock, 192 MB main memory
  - `thug01` through `thug32`: backend computers
    - UltraSPARC-IIe CPU, 650 MHz clock, 1 GB main memory
  - 100-Mbps switched Ethernet backend interconnection network
  - Aggregate 32 processors, 21 GHz clock, 32 GB main memory
- Hybrid
  - `tardis`: frontend computer
    - UltraSPARC-IIe CPU, 650 MHz clock, 512 MB main memory
  - `dr00` through `dr09`: backend computers
    - AMD Opteron 2218 dual-core CPUs (four processors), 2.6 GHz clock, 8 GB main memory
  - 1-Gbps switched Ethernet backend interconnection network
  - Aggregate 40 processors, 104 GHz clock, 80 GB main memory

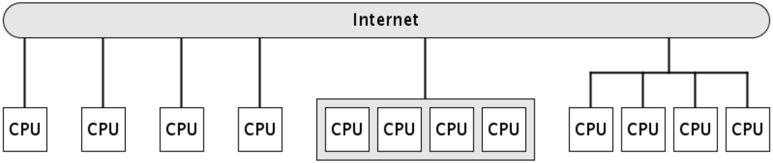
My New Toy (Manticore Project)

Dell PowerEdge R815

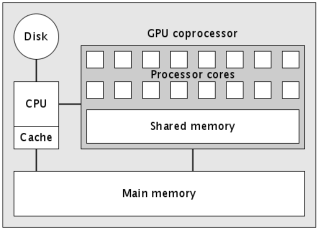


- Four AMD Opteron 6172 12-core CPUs (48 processors), 2.1 GHz clock, 128 GB main memory
- only $10,444.84

Grid Parallel Computers



GPGPU Parallel Computers



Parallel Programming

How do we *write* a program to make use of a parallel computer?

Use a standard programming language and generic OS kernel functions:

- SMP: C with `pthreads`, Java with `Thread` objects
- Cluster: C with sockets API, Java with socket classes

- Requires expertise in threads and/or sockets
- Much repetition accross parallel programs

Use a parallel programming library:

- OpenMP (SMP)
- MPI (Cluster)
- Parallel Java (SMP and Cluster)

Parallel Programming

How do we *design* a program to make use of a parallel computer?

Three patterns for designing parallel programs

- Result parallelism
- Agenda parallelism
- Specialist parallelism

Two additional patterns for implementing parallel programs

- Clumping (or slicing)
- Master-Worker

Steps for parallel program design

- Identify the pattern that best matches the problem.
- Take the pattern's suggested design as the starting point.
- Implement the design using the appropriate constructs.

A First Parallel Program

Primality tests via trial division