CSC 548: Parallel Systems

Instructor: Frank Mueller
TAs: Amir Bahmani, Arash Rezaei

Spring 2016

Questions?

- Read message board
- Post on message board
- Ask the TA
- Ask before class (if there is time)
- Ask after class
- Come by during office hours

Logistics

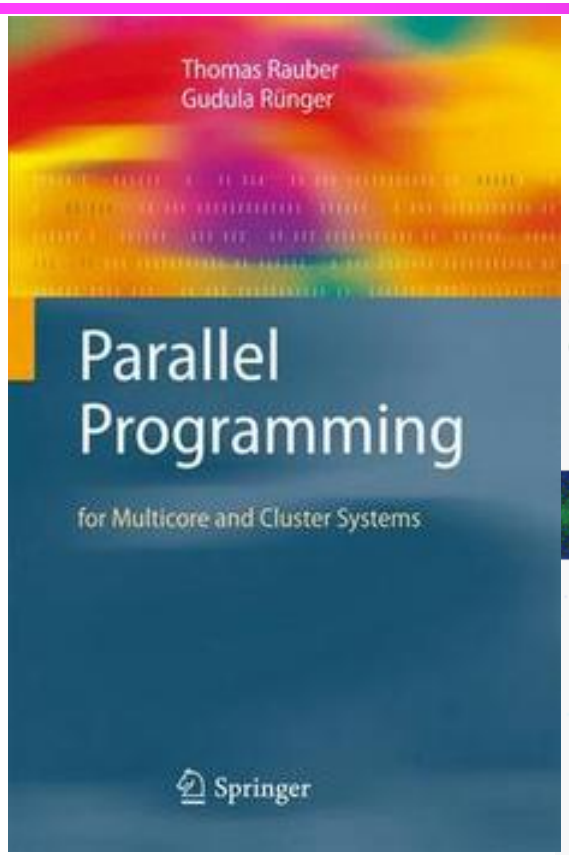
- Instructor: Frank Mueller
  - Office: EB2 3266
  - Office Hours: T 9:30am-10:30am
- TAs: Amir Bahmani, Arash Rezaei
  - Office: EB2 3226
  - Office Hours: M/W 12-1:30pm, Th/F 2-3:30pm
- More information
  - http://courses.ncsu.edu/csc548/lec/001

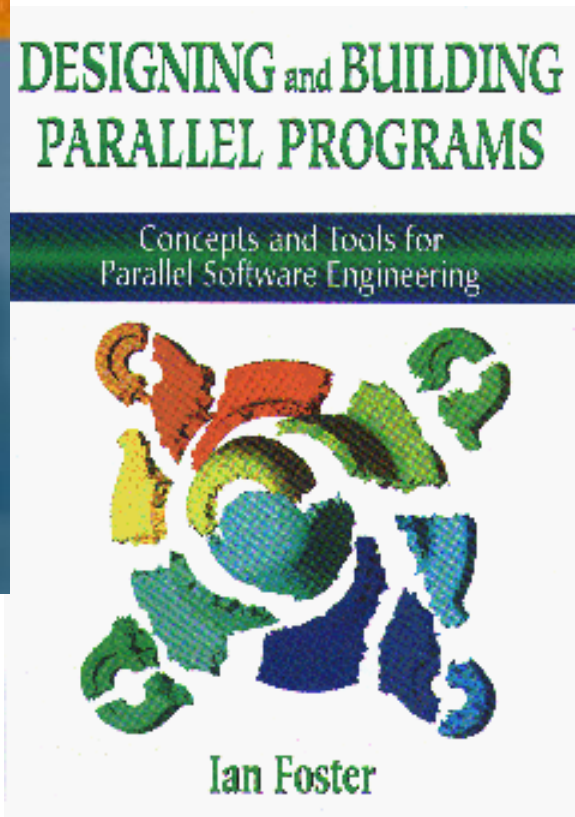
Course Overview

- Goals:
  - Parallel Distributed Computing: systems perspective
  - Compilation for Parallel Computing
- Structure:
  - Each major area:
    - Introduce basics quickly
    - Present and discuss papers
  - Assignments and project related to research

Online Textbooks

- Rauber/Rünger. Parallel Programming, Springer, 2010.
- Ian Foster. Designing and Building Parallel Programs, Addison-Wesley, 1995.
- Papers
  - Available on the web

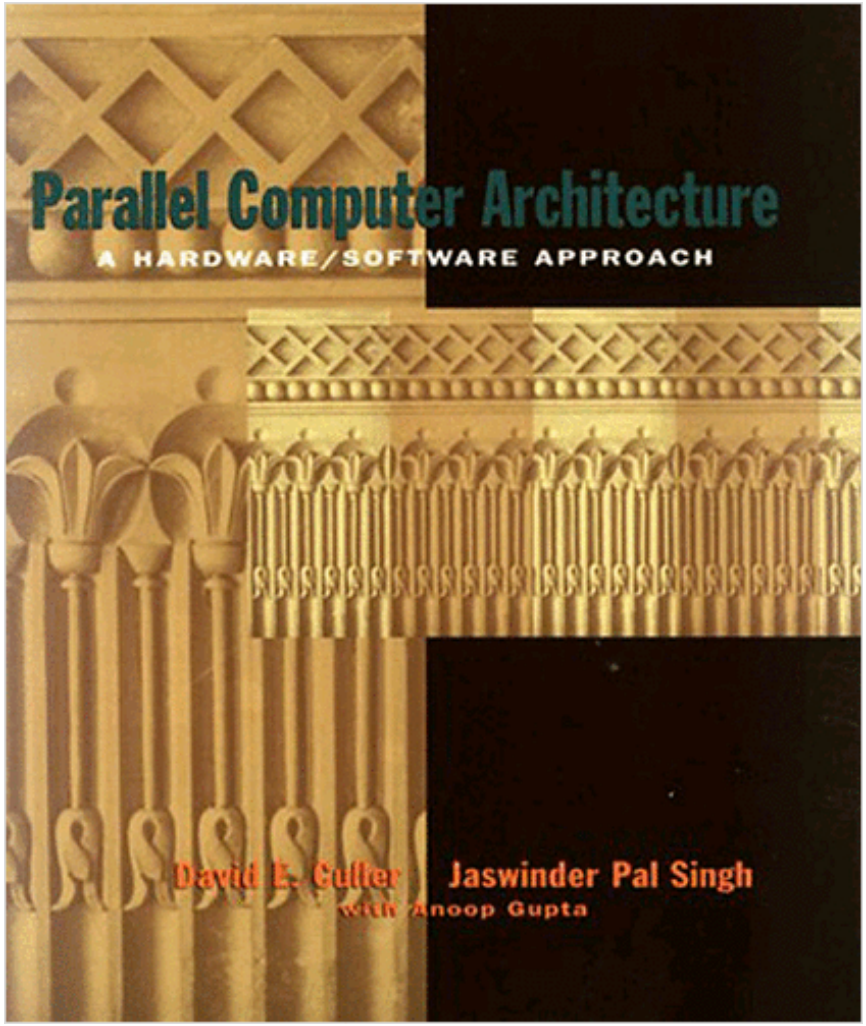




Textbooks - Recommended

- Recommended Reading:

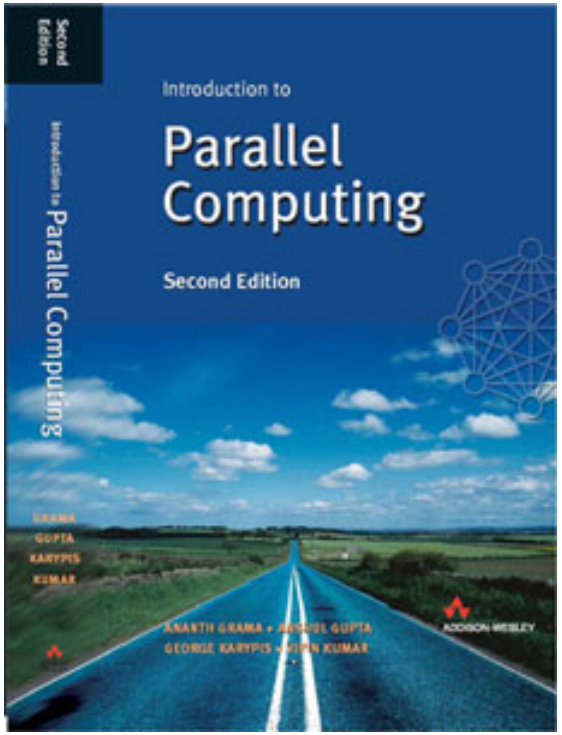
  Parallel Computer Architecture: A Hardware/Software Approach by David Culler et al., Morgan Kaufmann Publishers.



Textbooks - Recommended

- **Recommended Reading:**

  An Introduction to Parallel Computing: Design and Analysis of Algorithms (Ed. 2, Jan 2003) by Ananth Grama et al., Pearson Addison Wesley.



Topics

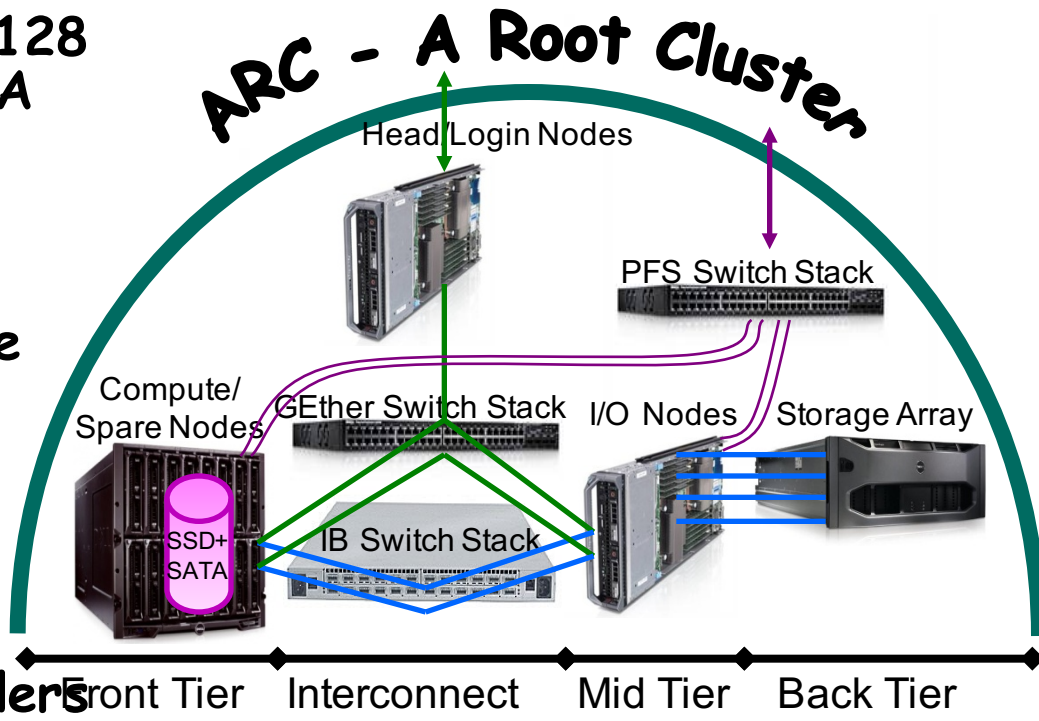
- Basics of parallel computing
  - Concepts
  - Architecture
  - Algorithms
- Parallel Programming
  - Message Passing
  - Shared-Memory Programming
  - Research topics
- Performance
  - Challenges
  - Choices
  - Limitations
  - Evaluation/analysis
- Research
  - Milestone papers (old)
  - Latest results (new)
  - Prepare for projects
  - Research beyond class

Programming Assignments / Project

- Goals
  - Learn to design, implement, and evaluate parallel programs
  - Improve systems programming skills: message passing, threads, synchroniza-tion, runtime systems, performance analysis, performance tuning, …
- Structure
  - Individual paper presentations (20 min.)
  - Small teams (2-3 students per team) and individual assignments
  - Written progress and project reports
- Facilities
  - Clusters: ARC, Henry2

ARC Cluster

- Beowulf-like cluster
- 108 Compute nodes
  - 16 cores, AMD Opteron 6128 2GHz, 32GB RAM, NVIDIA C2050/GTX480, 1TB HD, 120GB SSD
- Interconnects
  - 40Gb/s Infiniband fat tree
  - 1Gbps Ethernet switches
- Storage
  - 8TB NFS (default)
  - 36TB PVFS
- Gnu+PGI C/C++/Fortran compilers
  - MPI,OpenMP,CUDA,OpenACC



Scope and Assignments

- Parallel Programming
- Communication
- Synchronization
- Tools, Frameworks
- Projects

Course Requirements

- Prerequisites
  - Graduate OS and some architecture course
  - Good programming skills in C/C++ and UNIX
- What to expect
  - Readings and a presentation
  - Project
  - Several programming assignments
  - Midterm and Final exams or tests

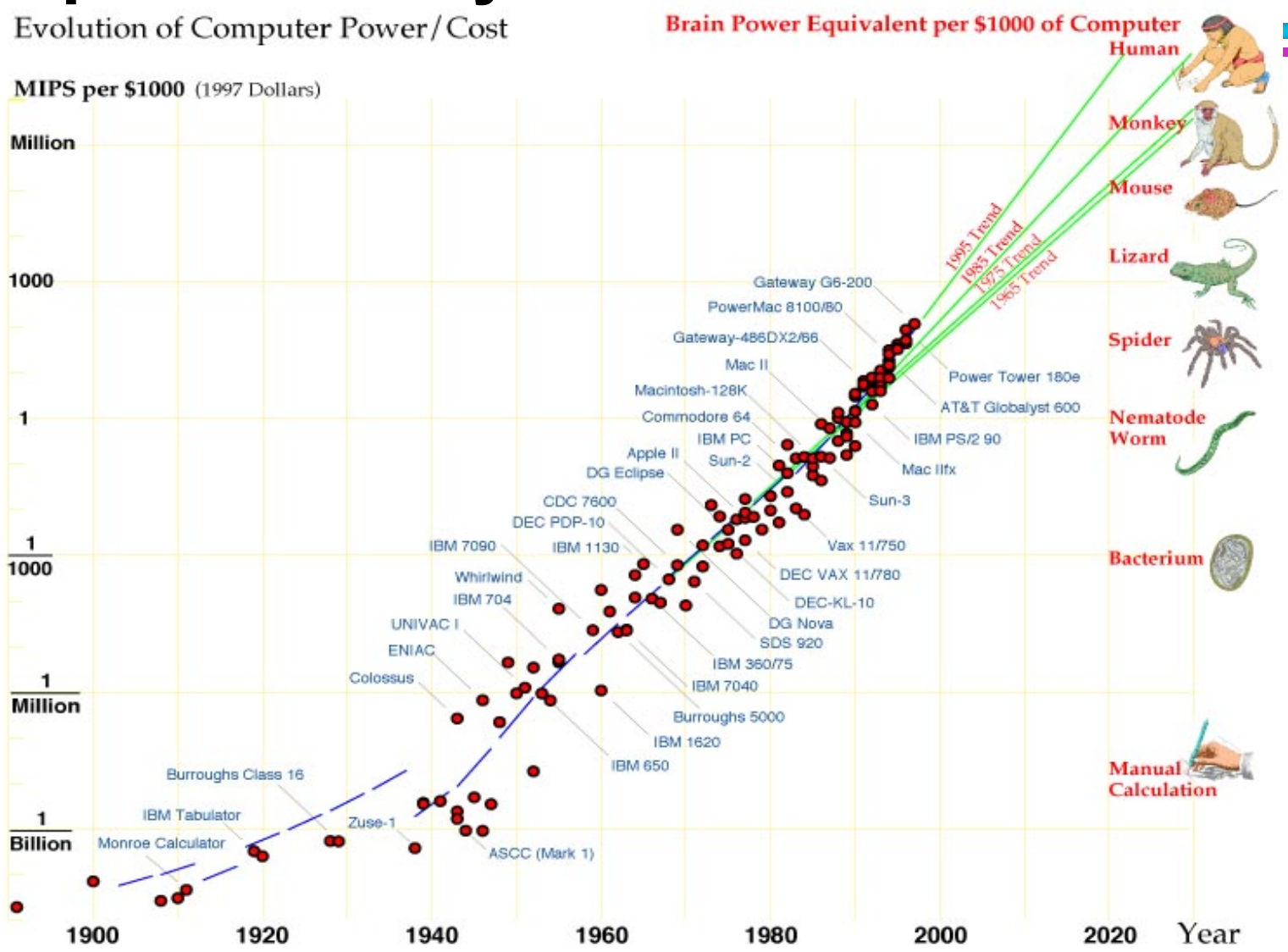
Grading

- Midterm 20%
- Final 20%
- Programming assignments and project 50%
- Presentation 10%
- On-line syllabus and policies

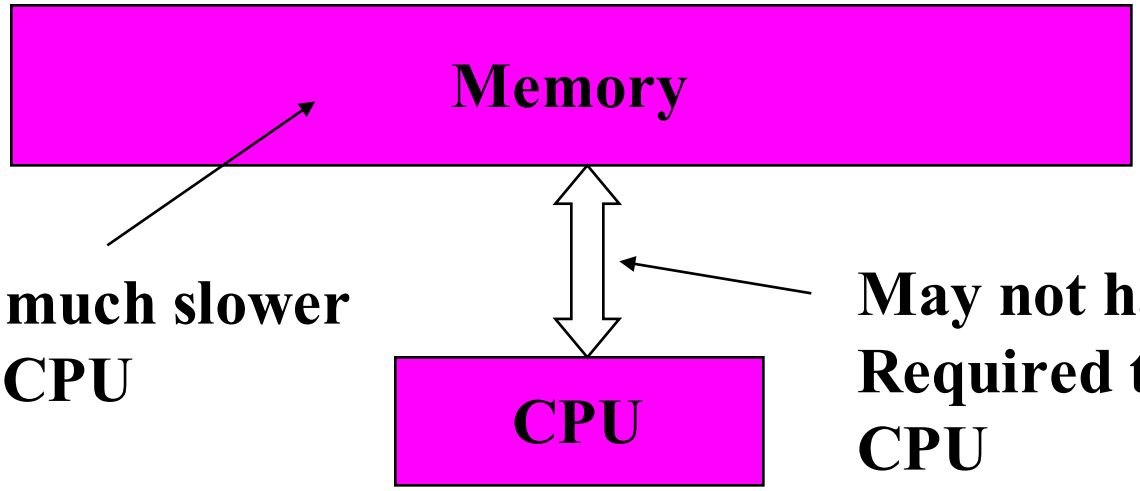
Today

- Why parallel?
  - Rauber Chapter 2/Foster Chapter 1
- Examples
- Classification
  - Hardware
  - Software
- Models
- Approaches

Computer History



Von Neumann Architecture



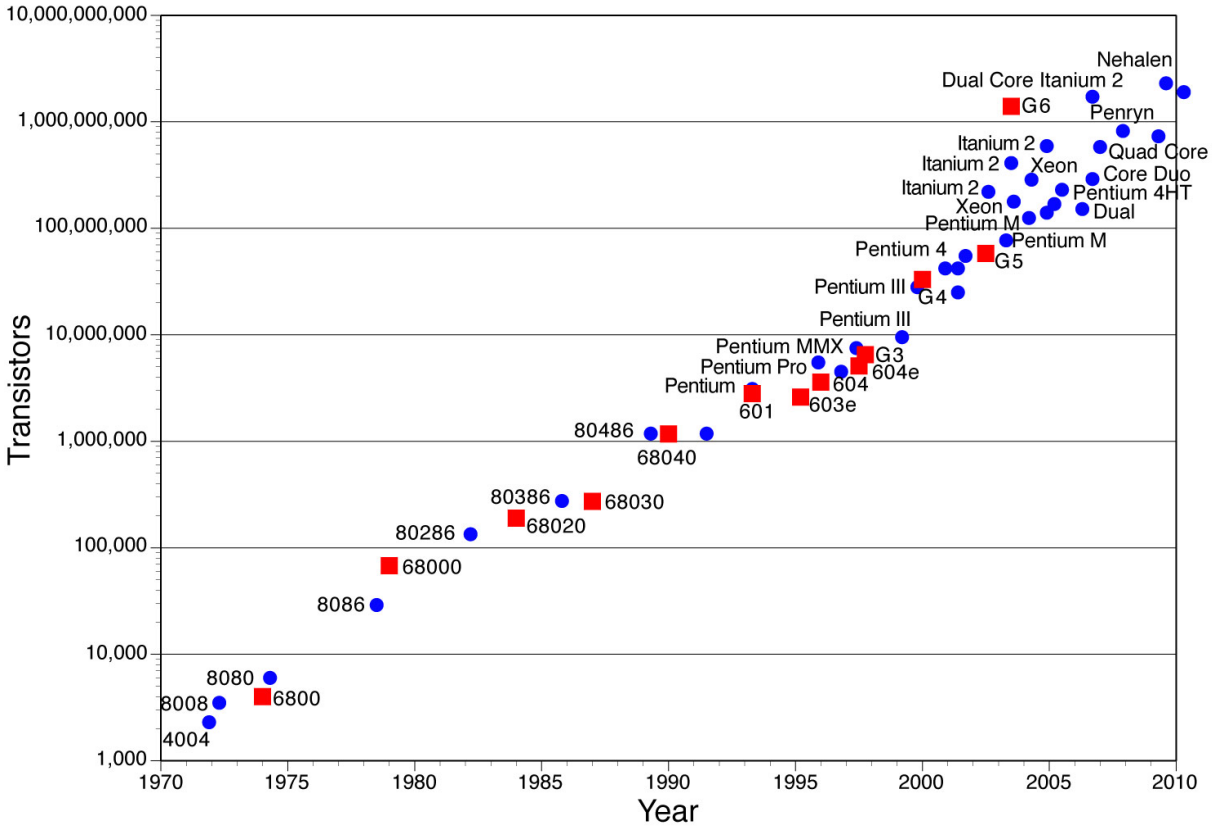
Typically much slower
Than the CPU

May not have bandwidth
Required to match the
CPU

- Some improvements have included
  - Interleaved Memory
  - Caching
  - Pipelining

Moore's Law

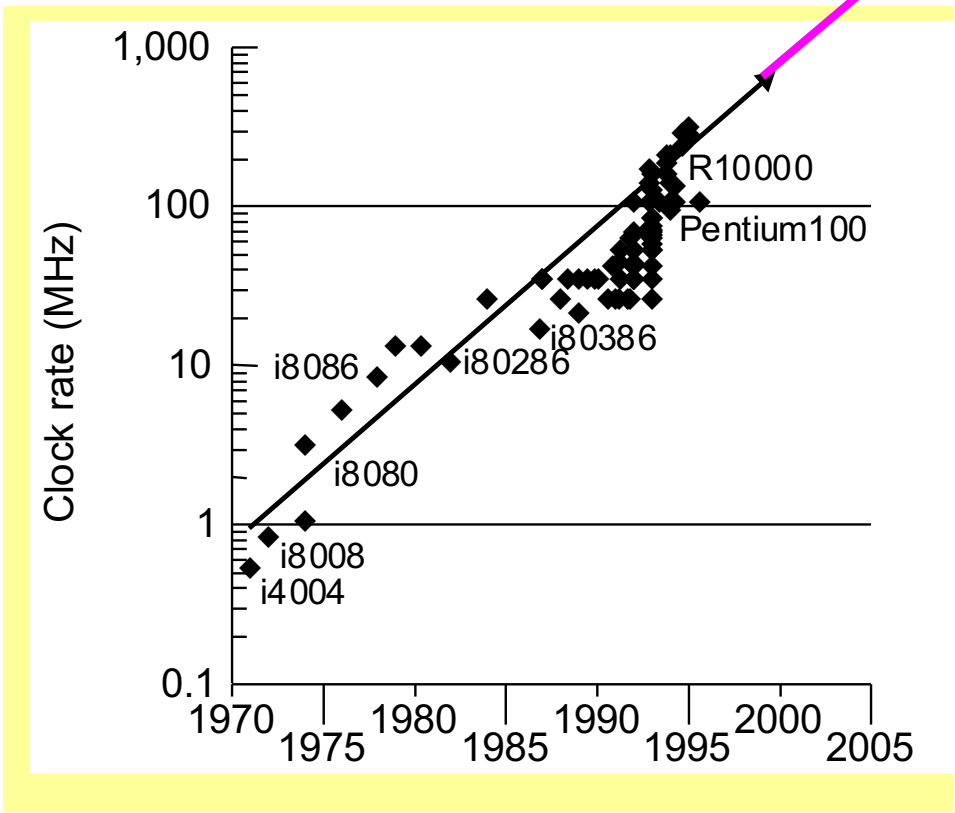
- Moore: Transistors double every 2 years



CSC 548

Moore's Law vs. Clock Frequency Growth

- Moore: Transistors double every 2 years
  - Frequency: 30% increase per year – up to ~2002 (max: ~3GHz)
  - Now: core count, cache sizes increase 30% per year

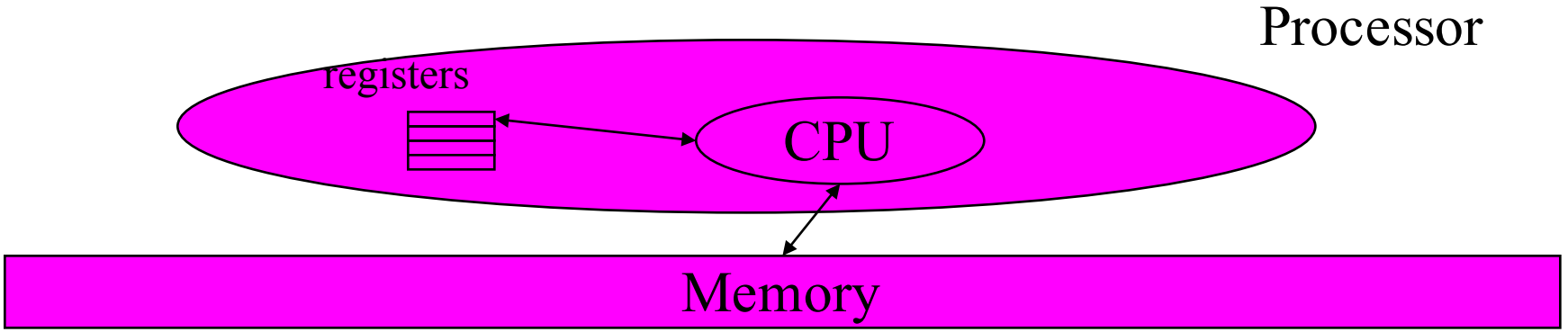


The Bottom Line

- Single processor: harder to extract ever more performance
  - But performance required by modern applications
- Practical constraints:
  - speed of light
  - heat dissipation and power consumption
- machines w/ multiple processing elements are future
  - Simultaneous Multi-Threading (SMT), aka Hyperthreading
  - Multi-core / Chip Multi-Processor (CMP)

Memory Hierarchy (1): Registers

- Processors faster than memory
  - can deal with data within processor much faster
- So, create some locations in processor for storing data
  - **registers**
  - Data movement instructions
    - Register ←→ memory
  - Computation instructions: arithmetic

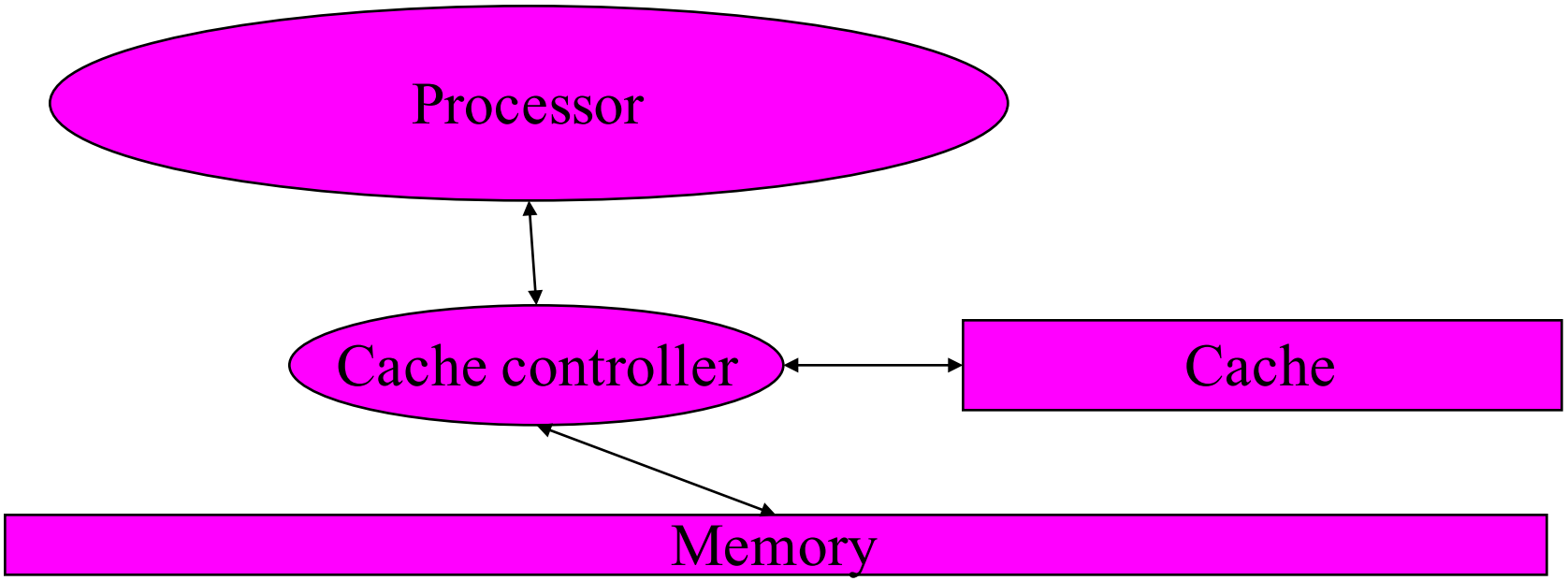


Memory Hierarchy (2): Caches

- processor still has to wait for data from memory
  - Although CPU is mostly executing register-only instructions
- Faster SRAM memory is available (although expensive)
- Idea: just like registers, put some more data in faster memory
  - Which data?
  - Principle of locality: (empirical observation)
    - Data accessed correlates with past accesses
    - → **spatially and temporarily locality**

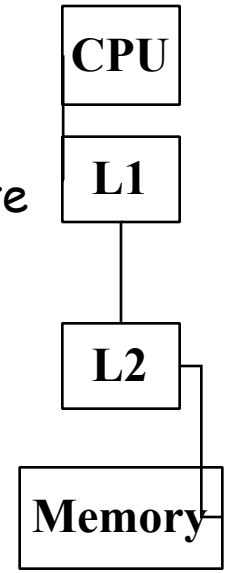
Caches

- Processor still issues load/store instructions
- but cache controller intercepts requests and,
  - if location is cached, resolves values from cache
- Data transfer between cache and memory not seen by processor

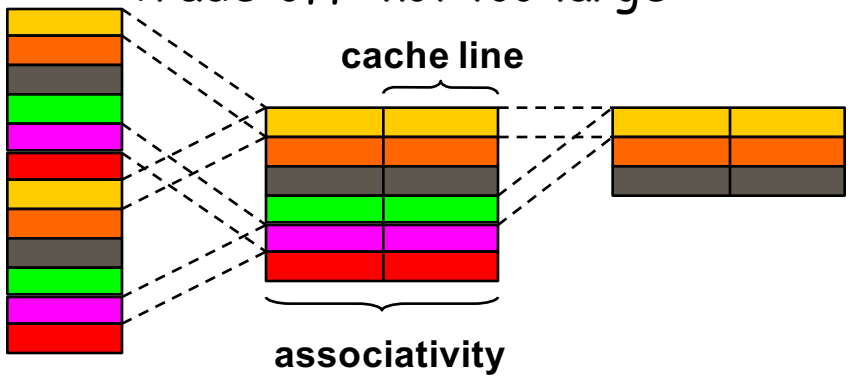


Cache Issues

- Multi-level caches
- A: Associative / replacement policy
- B: Block / line size
- C: Capacity / cache size
- Mapping phys./virtual
- Write thru/write back
- Alloc: on write/not on write
- 3 Types of misses: CCC
  - Cold: 1st referenced
  - Capacity: out of space
  - Conflict: assoc. too small



- Cache line / block:
  - bunch of data "at once" :
    - exploit spatial locality
    - block transfers faster
  - 64-128 byte cache lines
  - Trade-off: not too large



Cache line ~32-128
Associativity ~2-8

Cache Locality Example: 64 byte line size

1. Temporal: repeat same refs

   int a[36][16]; // 36*16*4 bytes

a[0][0],a[12][0],a[24][0],a[0][0],…

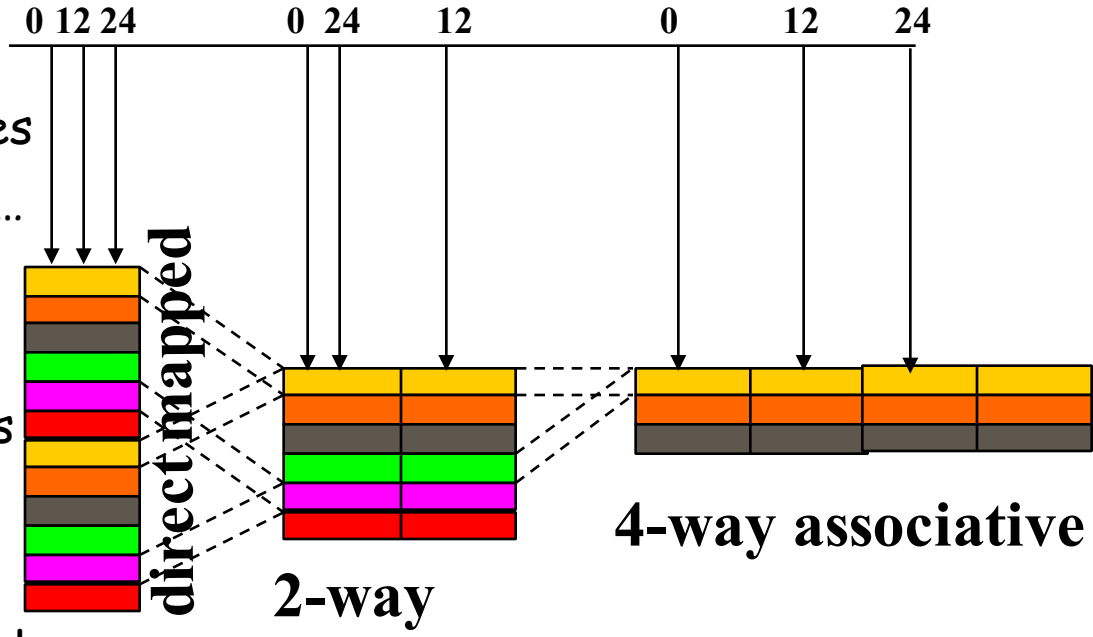
- Direct: conflict misses
- 2-way: conflict misses
- 4-way: 3 cold misses, then hits

Access all a[i][j] + again all a[i][j]

- Capacity misses on all a[i][0]
- Hits on a[i][j], j=1..15 → spatial

Access all a[0..11][j] + again same

- Cold misses on a[0..11][0] 1st time
- Hits a[0..11][0] after → temporal
- Hits on a[i][j], j=1..15 → spatial



2. Spatial: diff. refs close together

   int i, j, k; //assume: same line

Accesses: i, j, k, i, j, k….

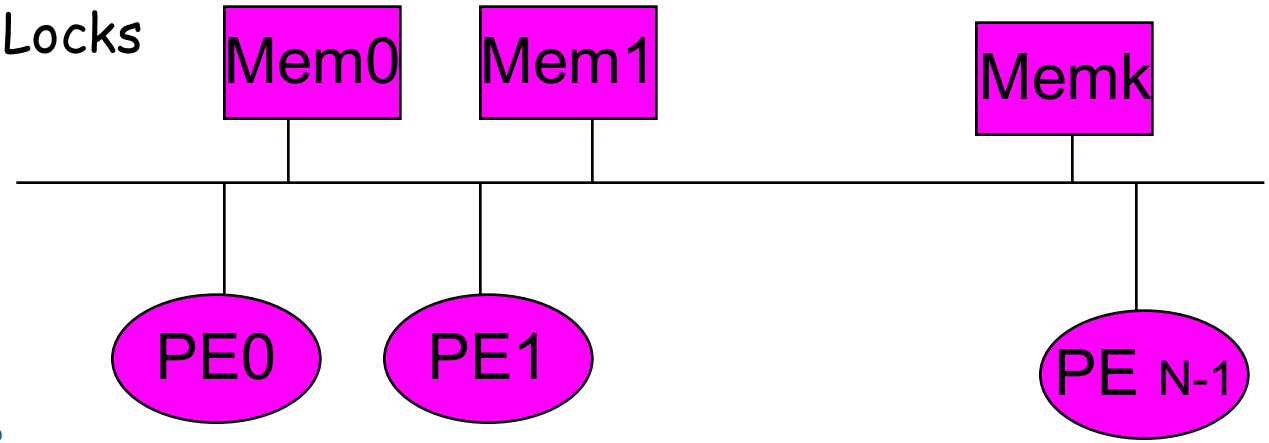
- 1st i: cold miss, other i/j/k: hits

Parallel Machines

- Several types of machines
  - Bus-based shared memory machines
  - Scalable shared memory machines
    - Cache coherent
    - Hardware support for remote memory access
  - Distributed memory machines
- Key issue: scalability

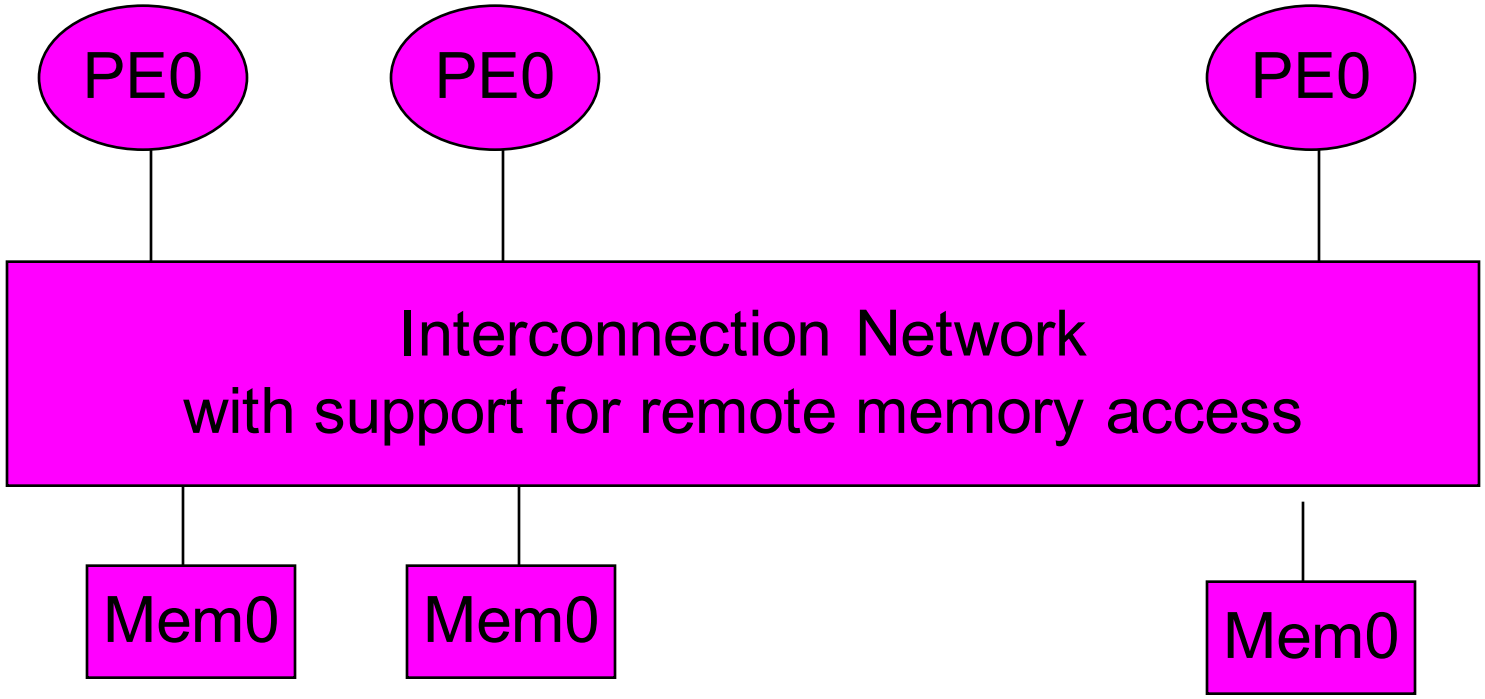
Bus Based Machines

- Any processor can access any memory location
  - Read and write
- Advantage?
  - Cheap connection
- Limiting factor?
  - Bus bandwidth
- Also, how do you deal with 2 processors modifying the same data?
  - Locks



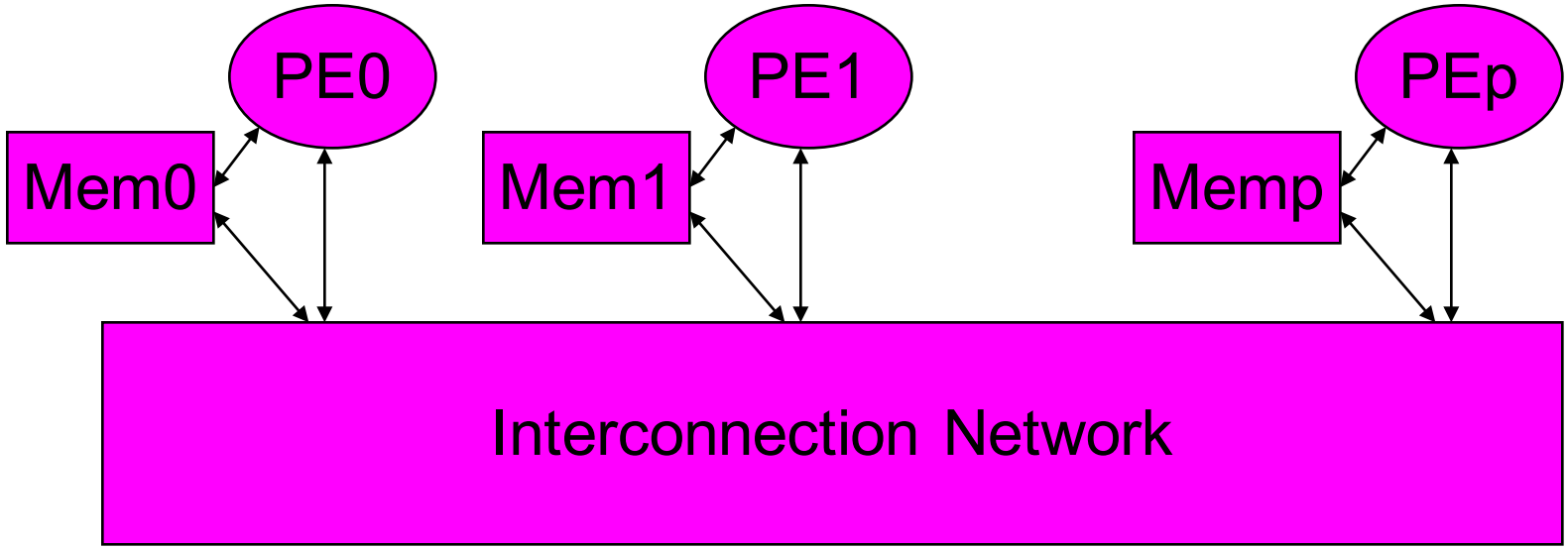
Scalable Shared Memory Machines

- *UMA*: Uniform Memory Access
  - But uniformly slow



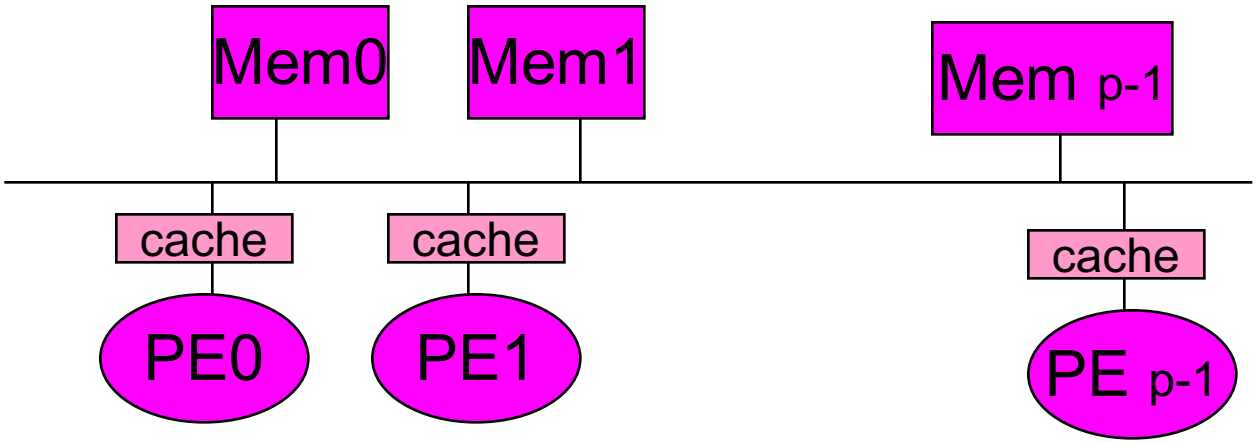
Distributed Memory Machines

- *NUMA*: Non-Uniform Memory Access →
  - Popular in contemporary machines (Hypertransport/QuickPath)



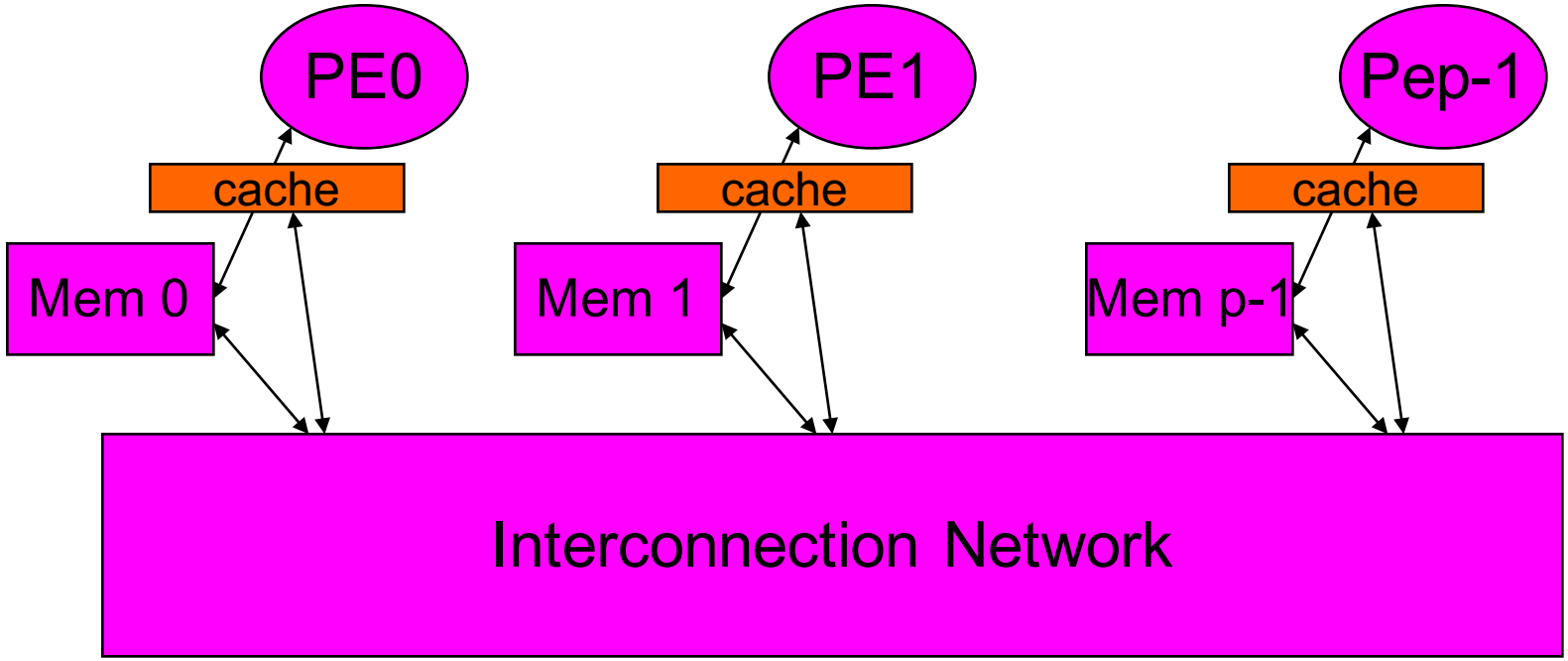
Introducing Caches into the Picture

- Now, we have more complex problems:
  - can't be fixed by locks alone
  - copy of same variables in 2 different caches
    → may contain different values
- Cache controller must do more



Distributed Memory Machines

- *CC-NUMA*: Cache-Coherent Non-Uniform Memory Access



Writing parallel programs

- Programming model
  - How should a programmer view a parallel machine?

- Parallel programming paradigms (more later):
  - Shared memory (shared address space) model
  - Message passing model

Concurrent Programming

- Operations that occur one after another, ordered in time
  - said to be sequential
- Operations that could be (but need not be) executed in parallel
  - concurrent

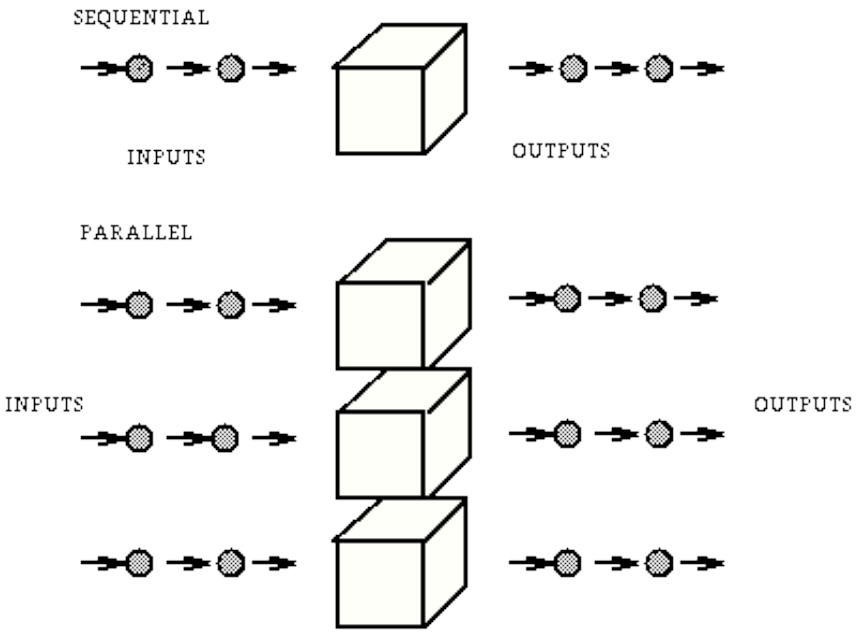
- 2 independent concepts:
  1. Concurrency in a language
  2. Parallelism in the underlying hardware are

Parallel Computing (Definition)

- A large collection of processing elements
  - can communicate
  - cooperates to solve large problems quickly
- A form of information processing
  - uses concurrent events during execution
    - both language and hardware support concurrency

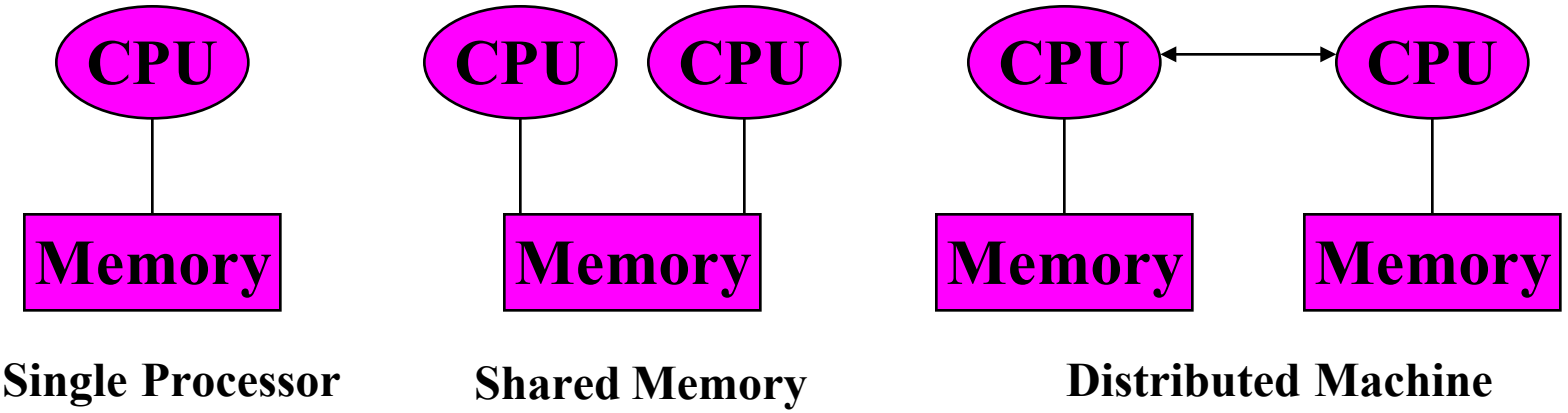
Parallelism

- If several operations can be performed simultaneously
  - total computation time reduced
- Here, parallel version has potential of being 3X faster



Parallelism in Hardware

- Dates back to the 50s
  - Data channels for controlling I/O devices



Parallel Architectures

- Unlike traditional von Neumann machines, there is no single standard architecture used on parallel machines
  - dozens of different parallel architectures have been built and are being used
  - But fewer custom designs lately, more commodity machines
  - Instead, more special communication interconnects
- Several classifications of different types of parallel machines exist
  - Flynn taxonomy is the most commonly used

Flynn's Model of Computation

- Any computer, whether sequential or parallel, operates by executing instructions on data
  - a stream of instructions (the algorithm) tells computer what to do.
  - a stream of data (the input) is affected by these instructions.
- Depending on whether there is one or several of these streams, Flynn's taxonomy defines four classes of computers

Flynn's Taxonomy

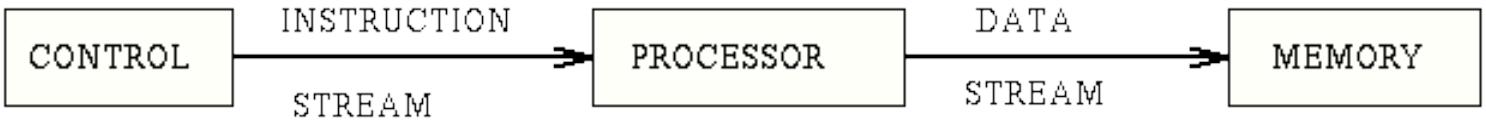
Data Streams

Instruction Streams

	Single	Multiple
Single	**S**ingle **I**nstruction **S**ingle **D**ata	**S**ingle **I**nstruction **M**ultiple **D**ata
Multiple	**M**ultiple **I**nstruction **S**ingle **D**ata	**M**ultiple **I**nstruction **M**ultiple **D**ata

SISD Computers

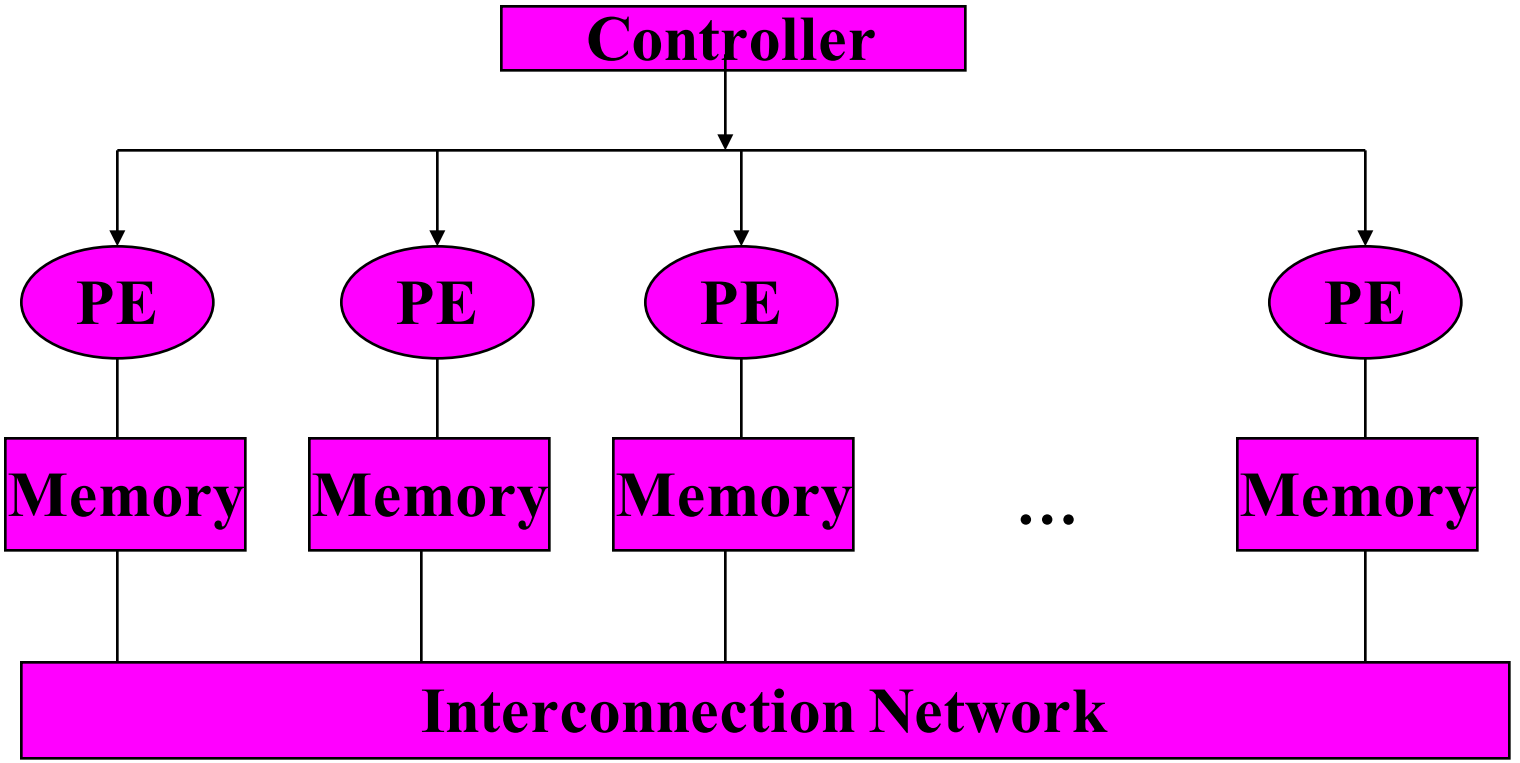
- Standard sequential computers
  - A single processing element receives a single stream of instructions that operate on a single stream of data
  - No parallelism here



SIMD Computers

- All processors operate under control single instruction stream
  - Processors can be selected under program control
- There are N data streams, one per processor
  - Variables can live either in parallel machine or scalar host
- Often referred to as data parallel computing

SIMD


Controller
PE
PE
PE
PE
Memory
Memory
Memory
...
Memory
Interconnection Network

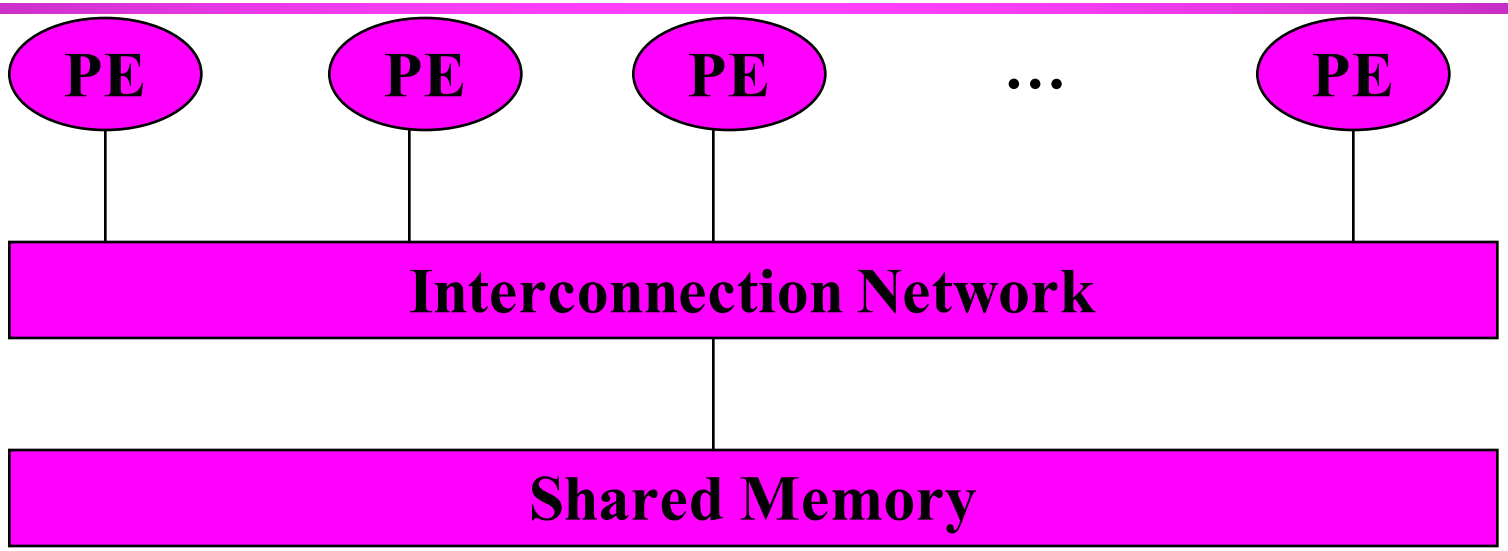
SIMD Algorithm

- Calculate number of heads in series of coin tosses
  - All processors flip a coin
  - If (coin is a head) raise your hand
- Note: there are better ways to count hands

MIMD Computers

- most general and most powerful of Flynn's taxonomy
  - N processors
  - N streams of instructions
  - N streams of data
- Each processor executes instruction stream
  - Processor can execute its own program on a different data
  - Processors operate asynchronously
    (doing different things on different data @ same time)

MIMD – Shared Memory

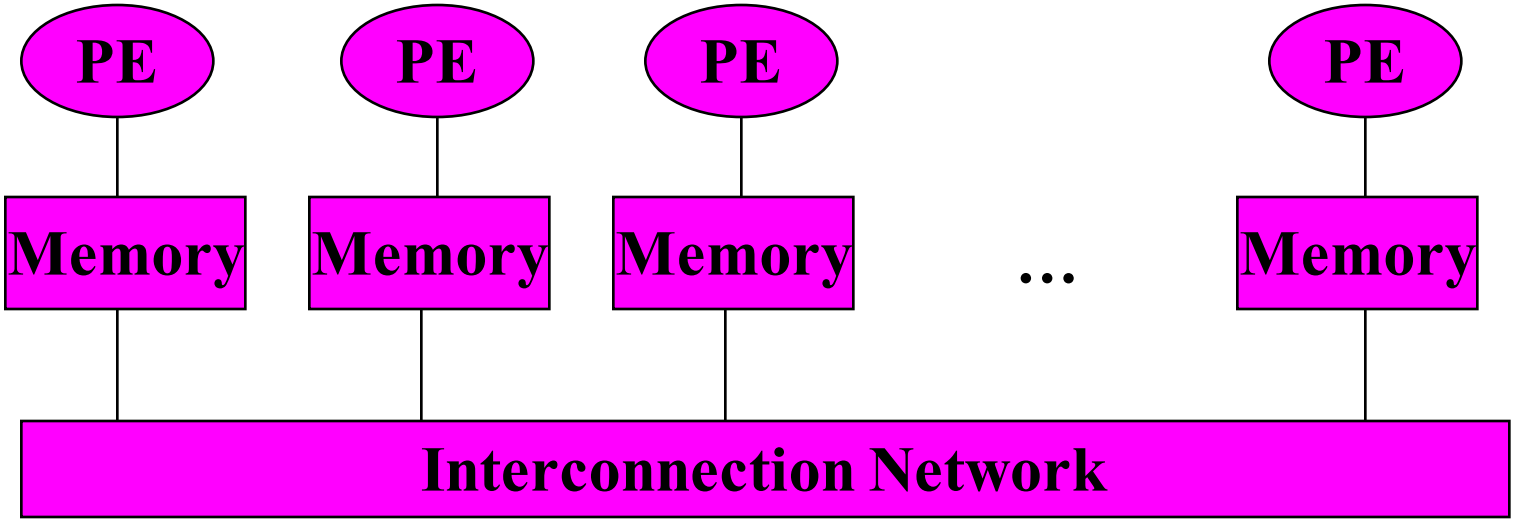


MIMD computers w/ shared memory

a.k.a

- Shared-memory multiprocessors (SMPs) or
- tightly coupled machines

MIMD – Message Passing



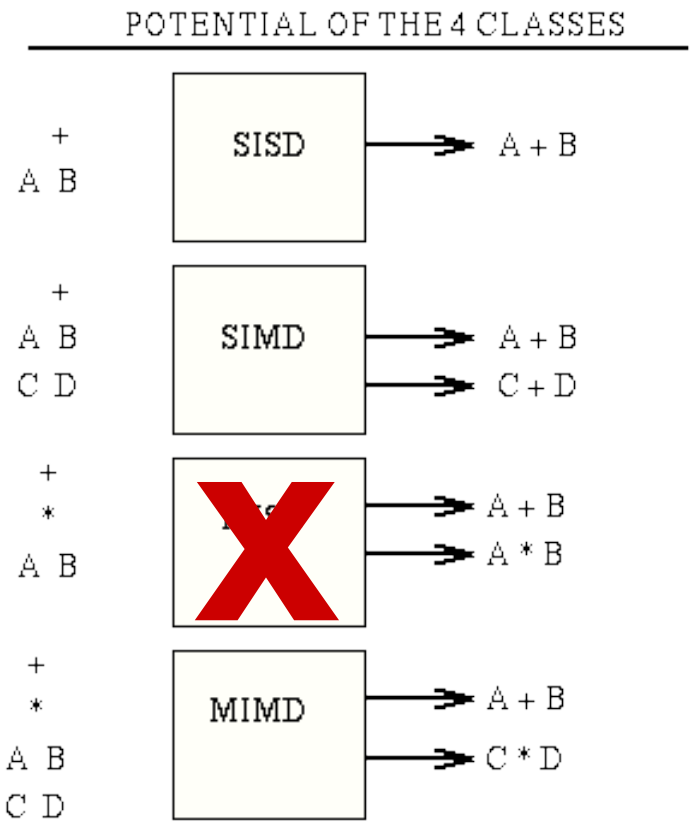
MIMD computers w/ interconnection network a.k.a.

- multicomputers or
- loosely coupled machines
- Multicomputers: sometimes referred to as distr. systems → incorrect
- Distributed systems: a network of computers
  - # processing units can be large but communication is typically slow

MIMD Algorithm

- Generate prime numbers from 2 to 20
  - Receive a number
  - This is your prime
  - Repeat
    - Receive a Number
    - If this number is not evenly divisible by your prime
      - then pass it to next processor

The 4 Classes



SPMD Computing

- Single Program Multiple Data
  - same program run on processors of a MIMD machine
  - occasionally, processors may synchronize
  - entire program executed on separate data →
    - possible that different branches are taken
    - leading to asynchronous parallelism
- SPMD came about as a desire to do SIMD like calculations on MIMD machines
  - SPMD is not a hardware paradigm, it is the software equivalent of SIMD
  - Clusters often use it

Parallel Problem Solving Style

- Synchronous (SIMD)
  - Same operation performed on all data points at same time
- Loosely Synchronous (SPMD)
  - Same operations performed by all processors, but not exactly the same time
  - Not synchronized at the computer clock cycle
    - Rather, "every now and then"
- Asynchronous (MIMD)
  - Every processor executes its own instruction on its own data

Types of Parallelism

- Overt
  - Parallelism is visible to the programmer
  - Difficult to do (right)
  - Large improvements in performance
- Covert
  - Parallelism is not visible to the programmer
  - Compiler responsible for parallelism
  - Easy to do
  - Small improvements in performance

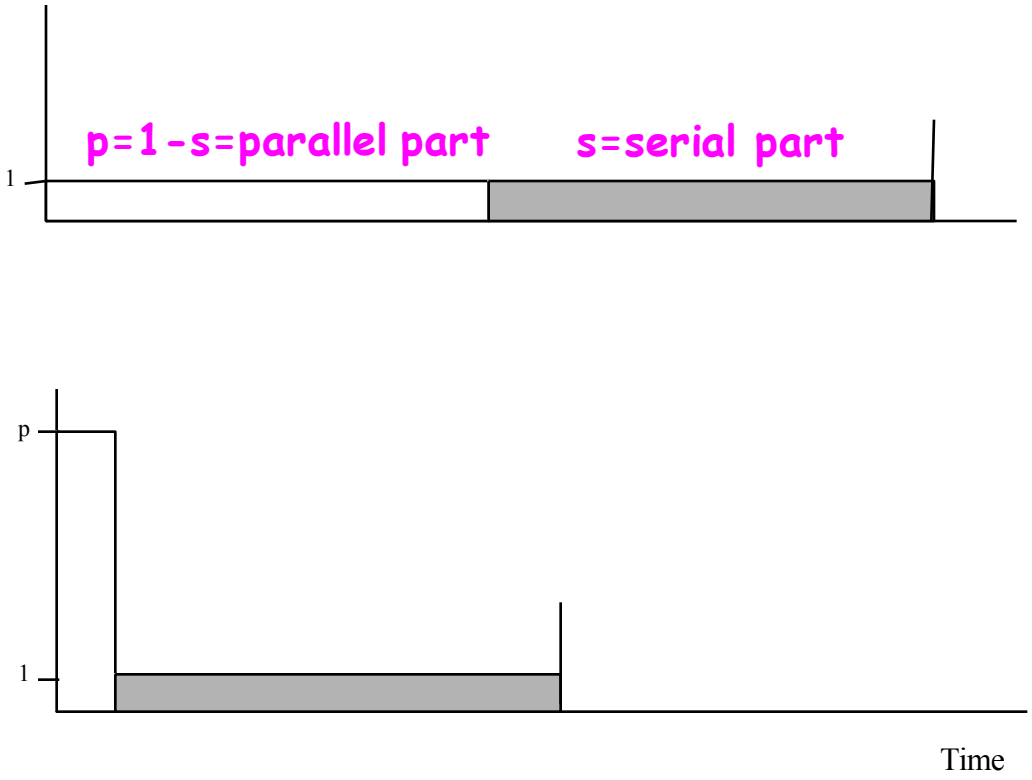
Why Parallel?

- Faster than sequential
- It depends:
  - Coordination
    - Synchronization
    - Communication
  - Scatter & gather
- Suitability of problem for parallelization
- Methodology for parallelization
  - Programming techniques
  - Hardware considerations
  - Compiler transformations

Task (Strong) Scaling: Amdahl's Law

- Given n processors, how long would parallel program take? (assumes same problem size / data set)
- If a fraction *s* of a computation is not parallelizable, then the best achievable speedup is

$$S = \frac{1}{s}$$

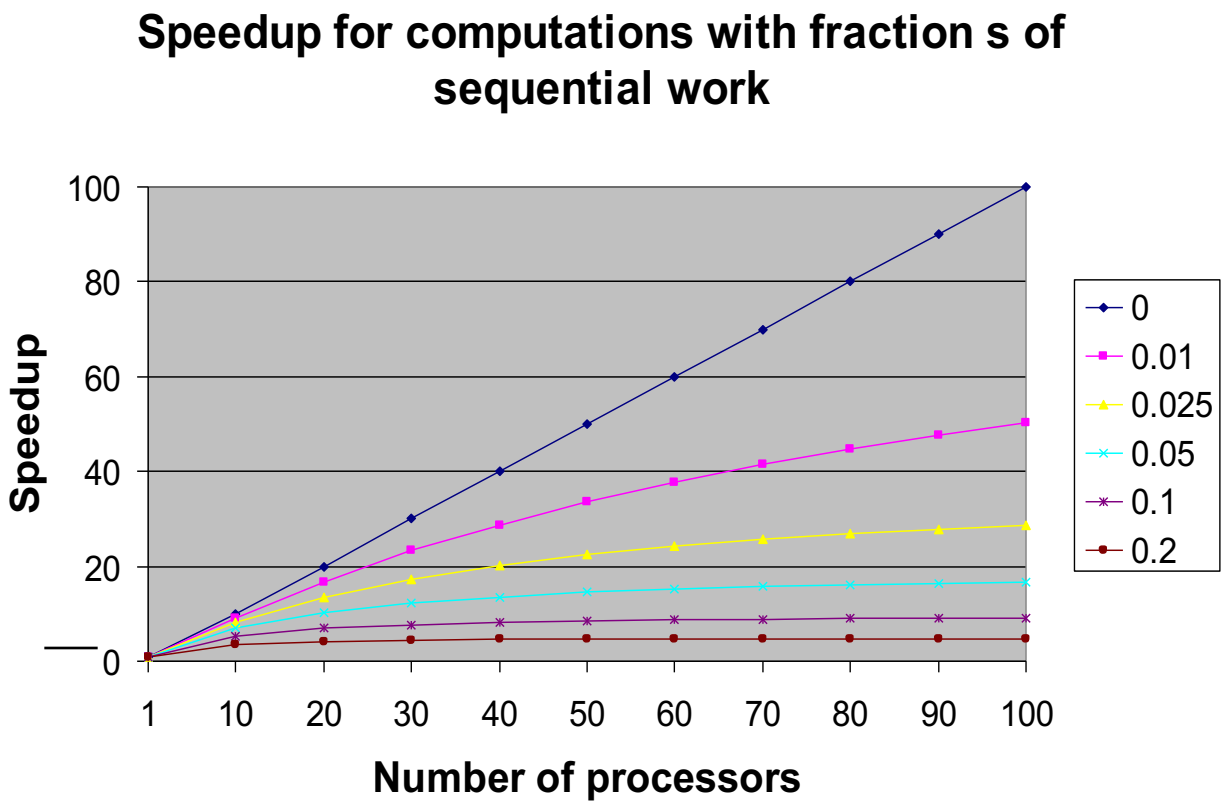


- Actually:

$$\lim_{n\to\infty} \frac{1}{s + \frac{1-s}{n}} = \frac{1}{s}$$

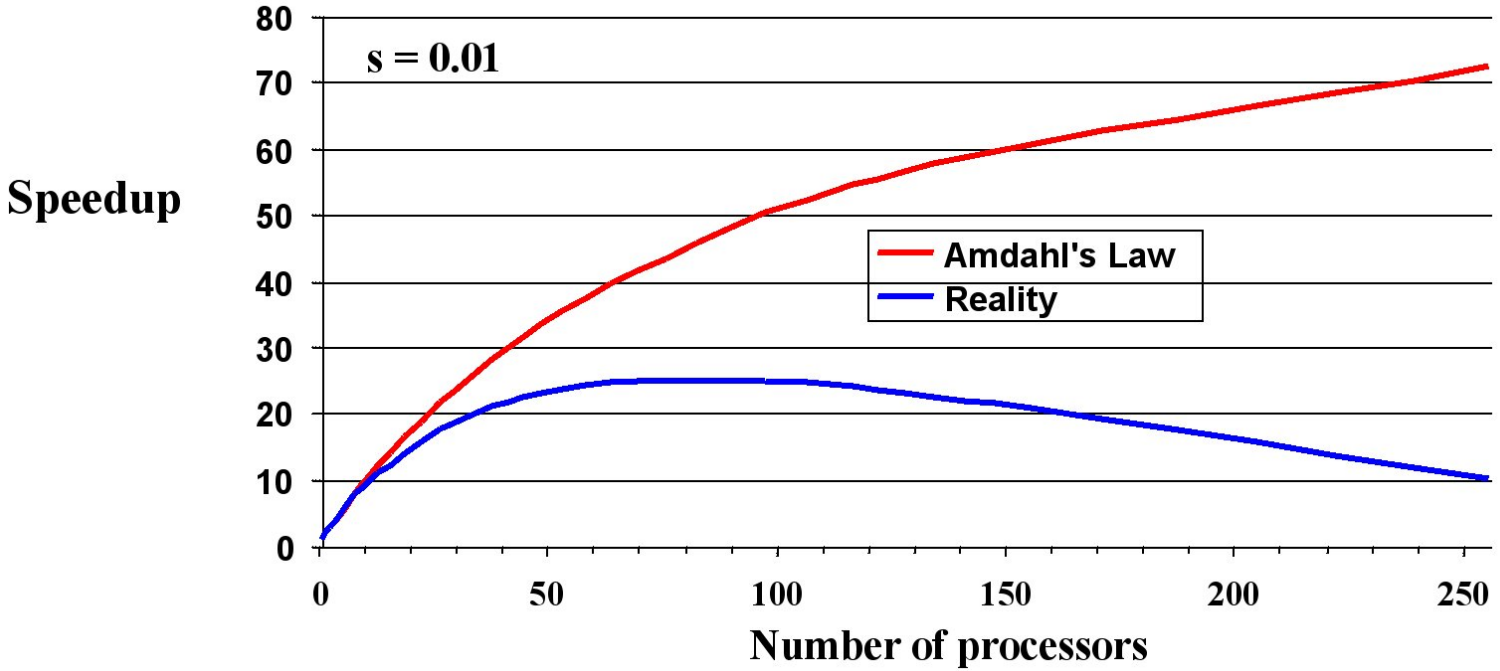
Amdahl's Law

$$S = \frac{1}{s}$$



Practical Limits: Amdahl's Law vs. Reality

- Amdahl's Law: theoretical upper limit on parallel speedup
  - assumes zero cost for speedup (e.g., zero costs for communication)
  - Reality: communication degrades of performance



Practical Limits: Amdahl's Law vs. Reality

In reality, Amdahl's Law limited:

- Communications
- I/O
- Load balancing (waiting)
- Scheduling (shared processors or memory)
- Algorithm
- Excess computation (redundancies)
  - Replication
  - Checkpoint/restart

Phenomenon: ***Super-linear speedup***

- Speedup > *n*
- Possible. Where?
- Extra hardware advantage exploited by parallelization
  - Better utilization of faster storage (cache, memory)
  - Exploratory decomposition

Problem (Weak) Scaling: Gustafson's Law

$$\text{Speedup} = \frac{s + p}{s + \frac{p}{n}} = \frac{1}{s + \frac{p}{n}} \quad \text{for } s+p=1$$

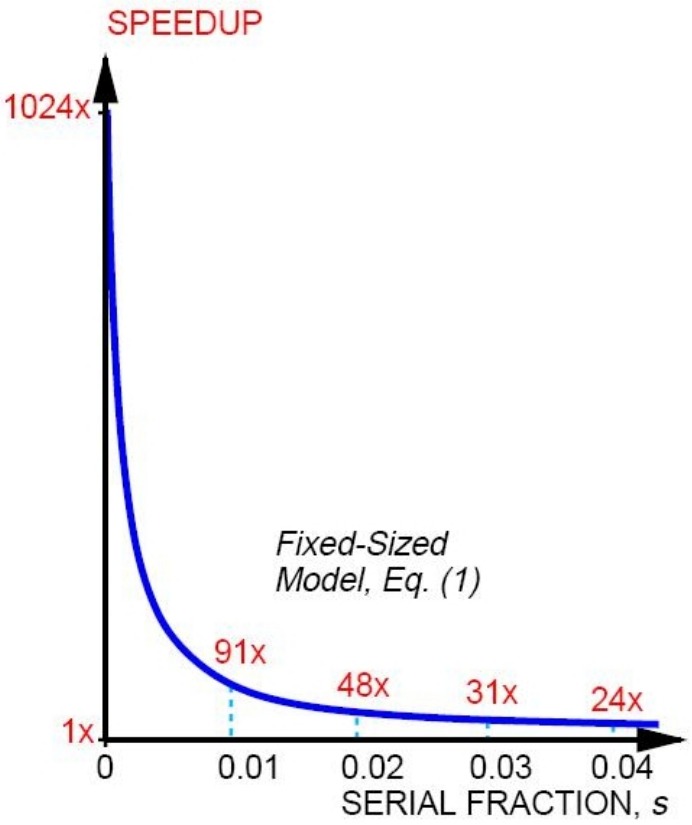
- Thesis: Amdahl's metric is unrealistic (Gustafson/Monty/Benner)
  - Given n processors, you don't run same problem size
  - Instead, scale up your problem size @ same rate as n
- "problem scaling" or "weak scaling"
- Given n processors, how long would serial program take?
  (now for different problem size / data set)
  - p' = parallel part w/ parallel (large) problem size

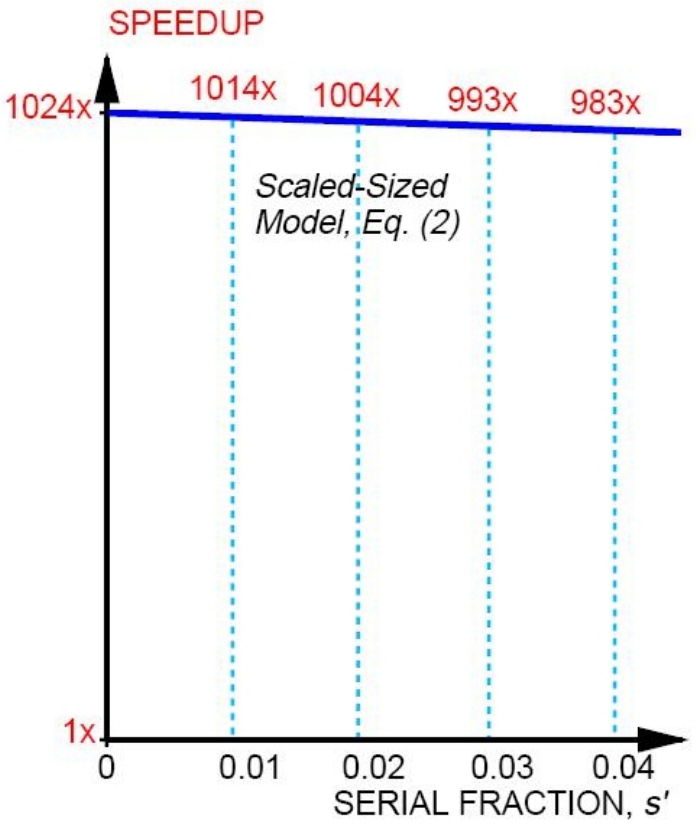
$$\text{Scaled speedup} = \frac{s' + p'n}{s' + p'} = n + (1-n)\, s' \quad \text{for } s'+p'=1$$

Scaled Speedup (Gustafson/Monty/Benner)

$$\text{Speedup} = \frac{1}{s + \frac{p}{n}}$$

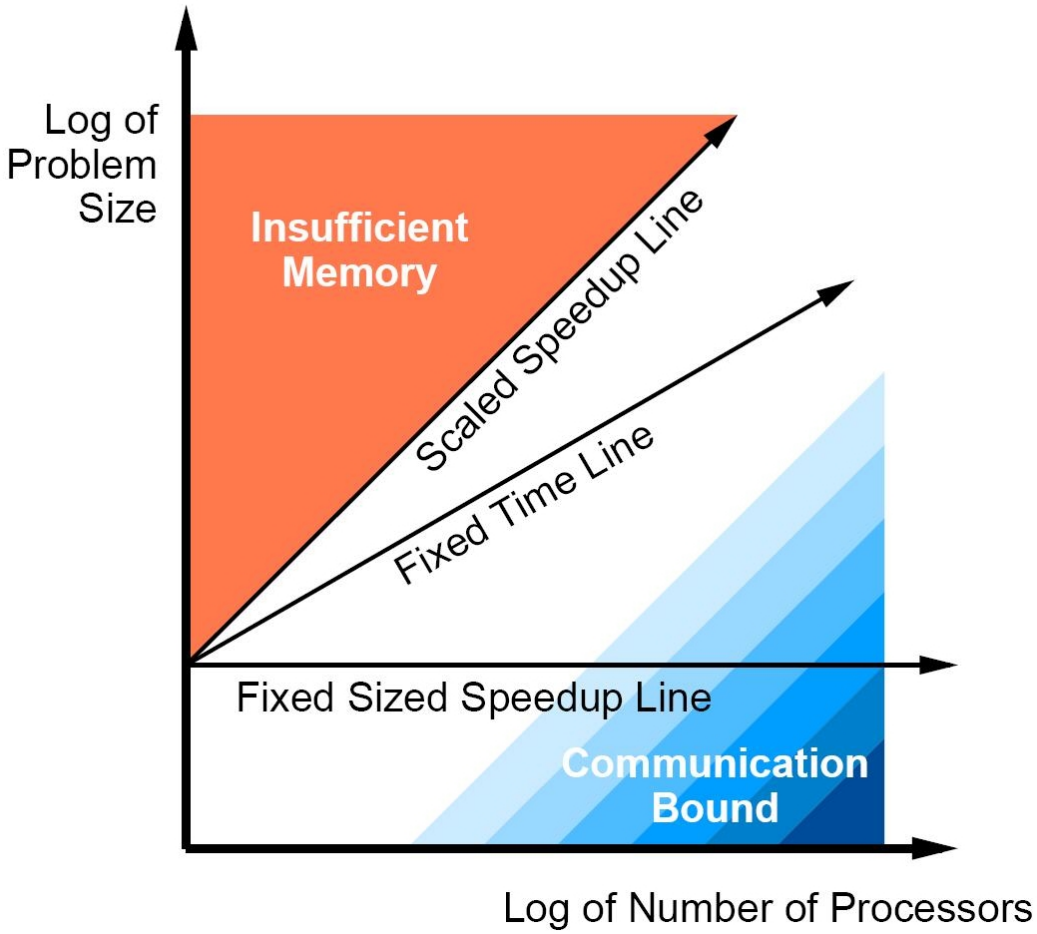
Scaled speedup
= n + (1-n) s'





Limits on Problem Scaling

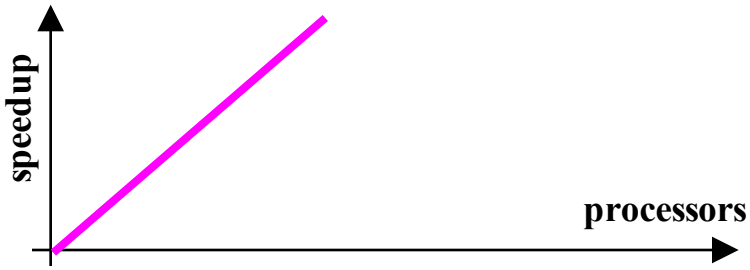
- Memory
- Communication



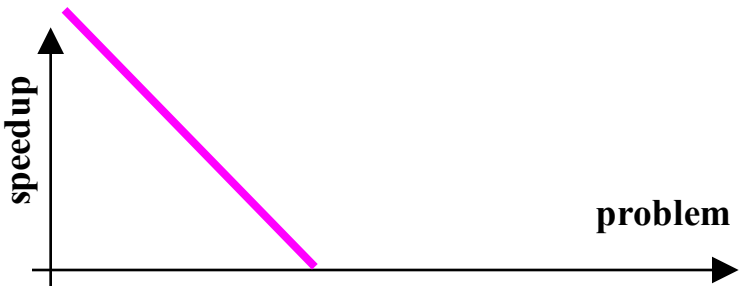
3 Types of Scaling

Shown here: ideal speedup

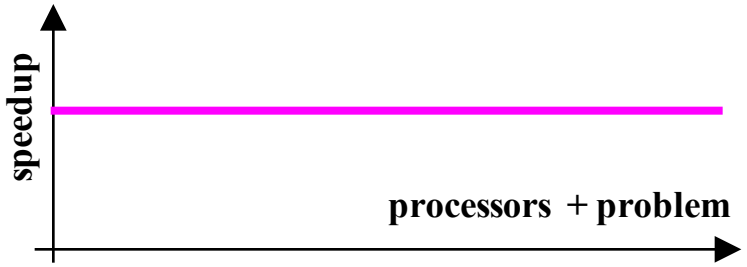
- Task (Strong) Scaling
  - Same problem size



- Problem-only Scaling
  - Same # processors



- Task+Problem (Weak) Scaling
  - Increase both @ same rate
  - Gustafson et al. call this problem scaling !!!



Other Considerations

- Writing effective parallel application is difficult!
  - Load balance is important
  - Communication can limit parallel efficiency
  - Serial time can dominate

- Is it worth your time to rewrite your application?
  - Do the CPU requirements justify parallelization?
  - Will the code be used just once?

Issues in Parallel Computing

- Design of parallel computers: Design so that
  - scales to large # of processor
  - supports fast communication
  - supports data sharing among processors
- Design of Efficient Algorithms:
  - Designing parallel algos different from designing serial algos
  - Significant amount of work is being done for numerical and non-numerical parallel algorithms
- Methods of Evaluating Parallel Algorithms: Given a parallel computer and a parallel algorithm, need to evaluate performance of resulting system
  - How fast is problem solved?
  - How efficiently are processors used?

Issues in Parallel Computing

- Parallel Computer Languages:
  - Parallel algos implemented using programming language
  - Language must be flexible enough to allow efficient implementation and must be easy to program in
  - Must efficiently use the hardware
- Parallel Programming Tools: Tools (compilers, libraries, debuggers, other monitoring or performance evaluation tools) must shield users from low level machine characteristics
- Portable Parallel Programs:
  - one of main problems w/ current parallel computers
  - programs written for one parallel computer require extensive work to port to another parallel computer

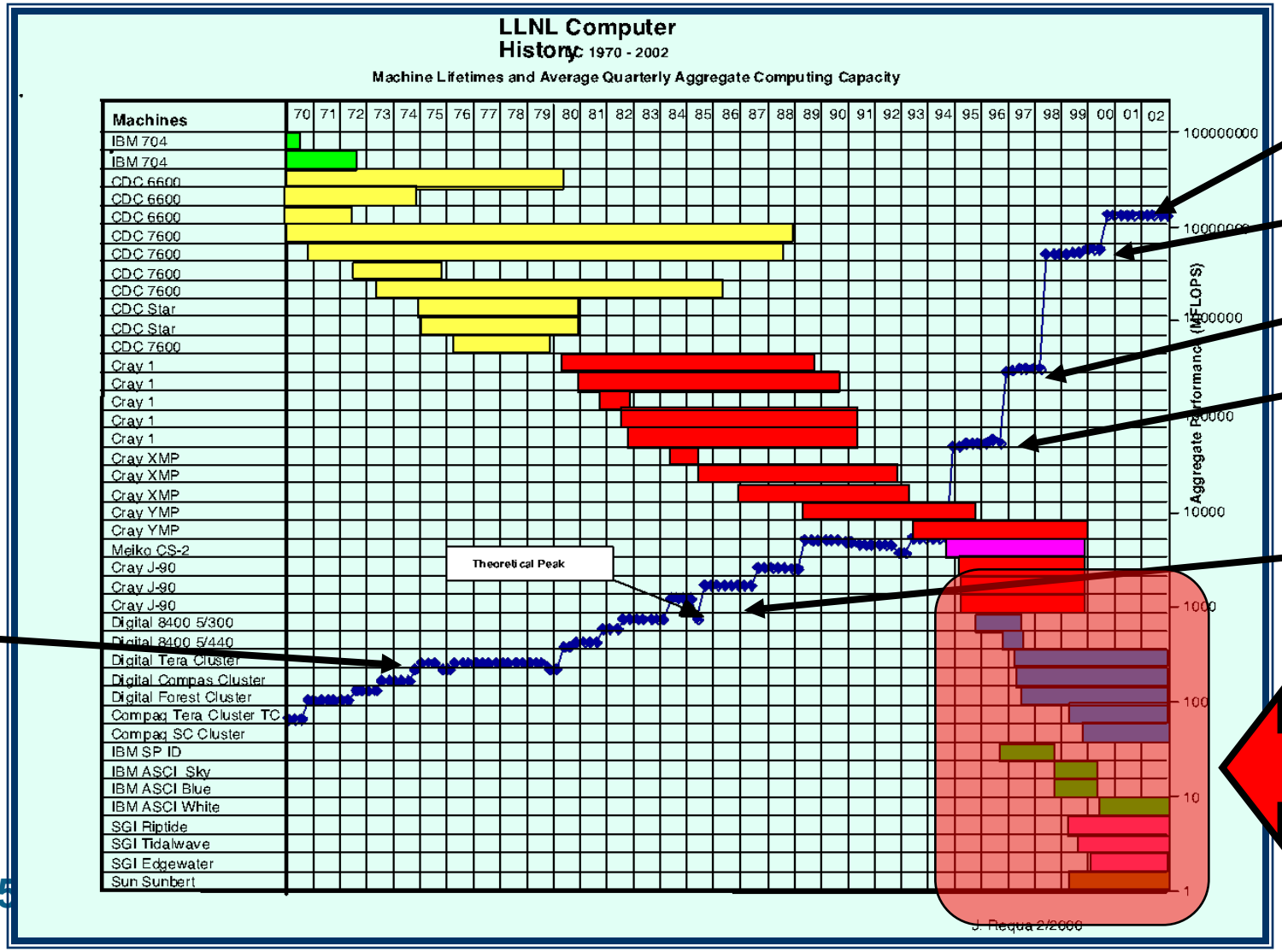
Issues in Parallel Computing

- Automatic Programming of Parallel Computers:
  - **design of parallelizing compilers which extract implicit parallelism from unparallelized programs**
  - **approach has limited potential for exploiting power of large parallel machines (and even for multi-cores)**
    - Good (not perfect) for SIMD / vectorization, e.g., SSE

Parallel Applications

- Scientific computing not only class of parallel applications
- Examples of non-scientific parallel applications:
  - Data mining
  - Real-time rendering
  - Distributed servers

LLNL Computer History (log plot)



CSC 5

Cluster for HW1

- ARC cluster in our department
- Accounts have been created
  - No account sharing
  - SSH to arc.csc.ncsu.edu
- ARC: Distributed memory machine
  - Commodity component cluster
  - Message passing programming
    - Plus limited shared memory programming
  - See HW1 page for link to manuals

Access ARC

- HOWTO: http://moss.csc.ncsu.edu/~mueller/cluster/arc
- Access: Need to be logged in to some ncsu.edu machine first!
  - then connect to login node (via head): ssh arc.csc.ncsu.edu
- Job queue managed with Maui/Torque
  - Submit your program with a job script: qsub …
  - Or run interactively: qsub -I …