

Lecture 1

Course Introduction

Welcome to CSE 262!

- Your instructor is Scott B. Baden
- Office hours (week 1)
 - Tues/Thurs 3.30 to 4.30
 - Room 3244 EBU3B

Content

- Our home page is
<http://www-cse.ucsd.edu/classes/sp11/cse262>
- All class announcements will be made on-line so check this web page frequently
- Moodle
- One recommended text:
Programming Massively Parallel Processors: A Hands-on Approach, by David Kirk and Wen-mei Hwu, Morgan Kaufmann Publishers (2010)
- Books (soon) on reserve in the S&E library
- Useful information on-line
<http://www-cse.ucsd.edu/users/baden/Doc>

Course Requirements

- Assignments: 25%
 - Early in the quarter
 - Programming, paper and pencil
 - To be done individually
- In class presentations (2 × 30 minutes): 15%
- Research Project: 60%
 - Weekly progress reports including in class presentations
 - Teams of 1 or 2
 - Final presentation during week 10
 - Report due Friday June 3, 2011 at 5pm

Course 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:

cseweb.ucsd.edu/classes/sp11/cse262-a/Policies.html

Course Overview and Background

- Latest trends in solving computationally intensive problems on parallel computers
 - Historical retrospective
 - Trends
 - Software and hardware
- Background
 - Graduate standing
 - Prior experience in parallel computation

Background markers

- C/C++ Java Fortran?

- Abstract base class

- Navier Stokes Equations

$$\nabla \cdot u = 0$$

- Sparse factorization

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot v) = 0$$

- TLB misses

- RPC

- Multithreading

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

- MPI

- CUDA, GPUs

Topics

- Computing with Graphical Processing Units (GPUs)
 - Advanced performance programming
 - Application studies
- Computing in the large
 - 10^4 to 10^5 processors and more (exascale)
 - Latency tolerance and communication avoidance
- Support
 - Programming languages and translators
 - Run time
- Irregular applications

Testbeds

- GPUs (UCSD)
 - lilliput (Tesla)
 - cseclass01 & 2 (Fermi)
- Scalable systems
 - Trestles.sdsc.edu: 10,368 cores
 - Kraken.nics.tennessee.edu: 99,072 cores

What is parallel processing ?

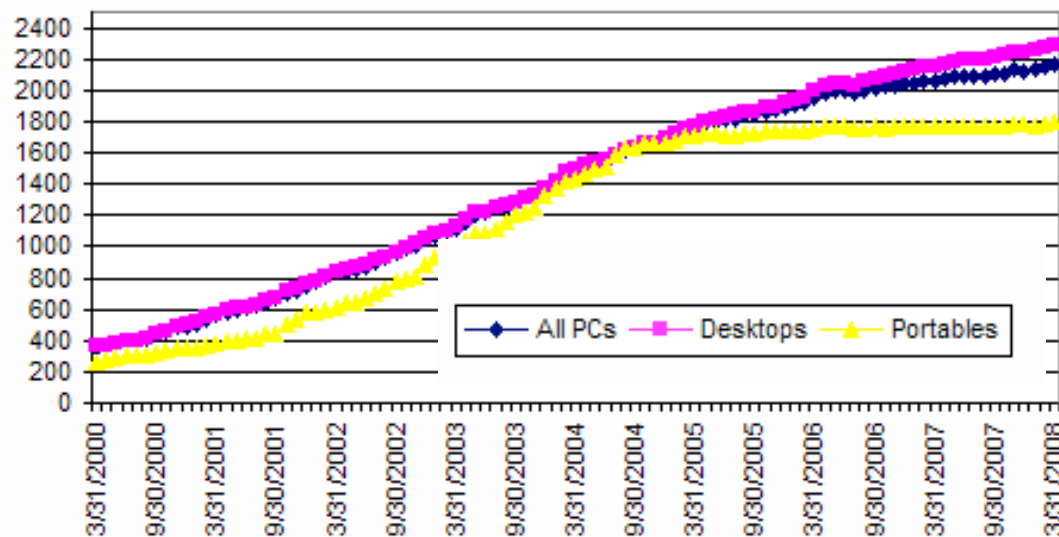
- Decompose a workload onto *simultaneously executing physical resources*
- Multiple processors co-operate to process a related set of tasks – *tightly coupled*
- Improve some aspect of performance
 - Speedup: 100 processors run $\times 100$ faster than one
 - Capability: Tackle a larger problem, more accurately
 - Algorithmic, e.g. search
 - Locality: more cache memory and bandwidth
- Virtual or physical
- Reliability more of an issue at the high end or in critical applications

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” or “Grid” computing, large amounts of storage
 - Looser, coarser grained communication and synchronization
- May or may not involve separate physical resources, e.g. multitasking “Virtual Parallelism”

Why is parallel computation inevitable?

- Physical limits on processor clock speed and heat dissipation
- A parallel computer increases memory capacity and bandwidth as well as the computational rate



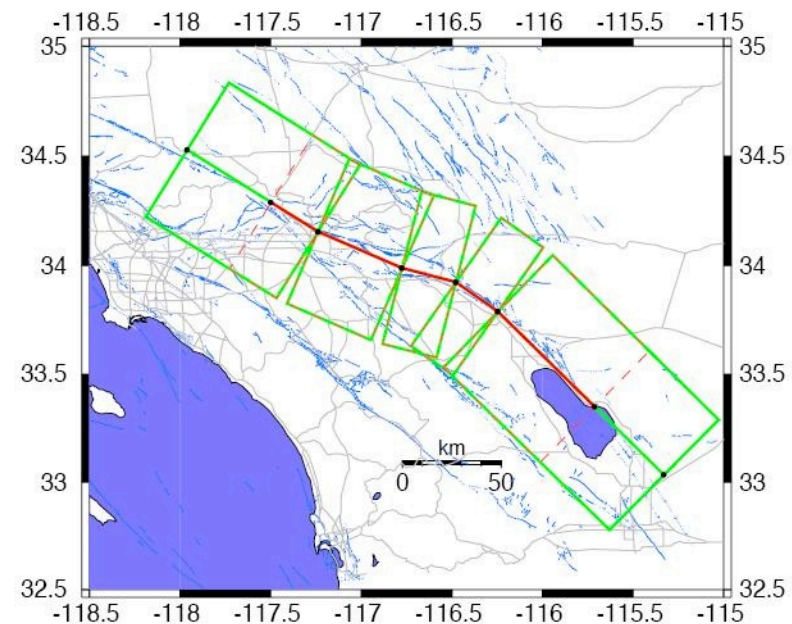
Average CPU clock speeds <http://www.pcpitstop.com/research/cpu.asp>



Nvidia

A Motivating Application - TeraShake

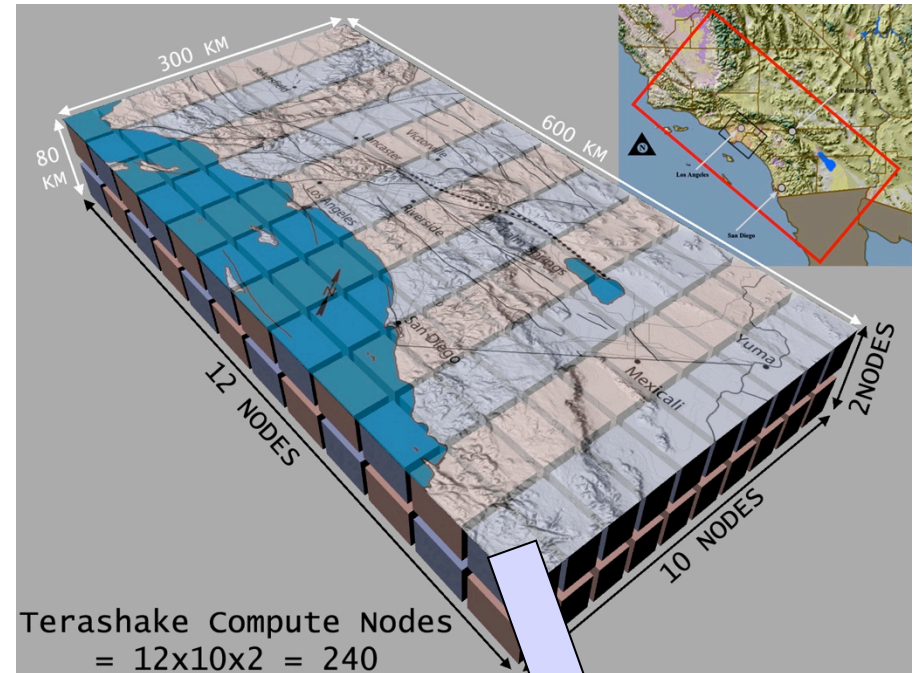
Simulates a 7.7 earthquake along the southern San Andreas fault near LA using seismic, geophysical, and other data from the Southern California Earthquake Center



epicenter.usc.edu/cmeportal/TeraShake.html

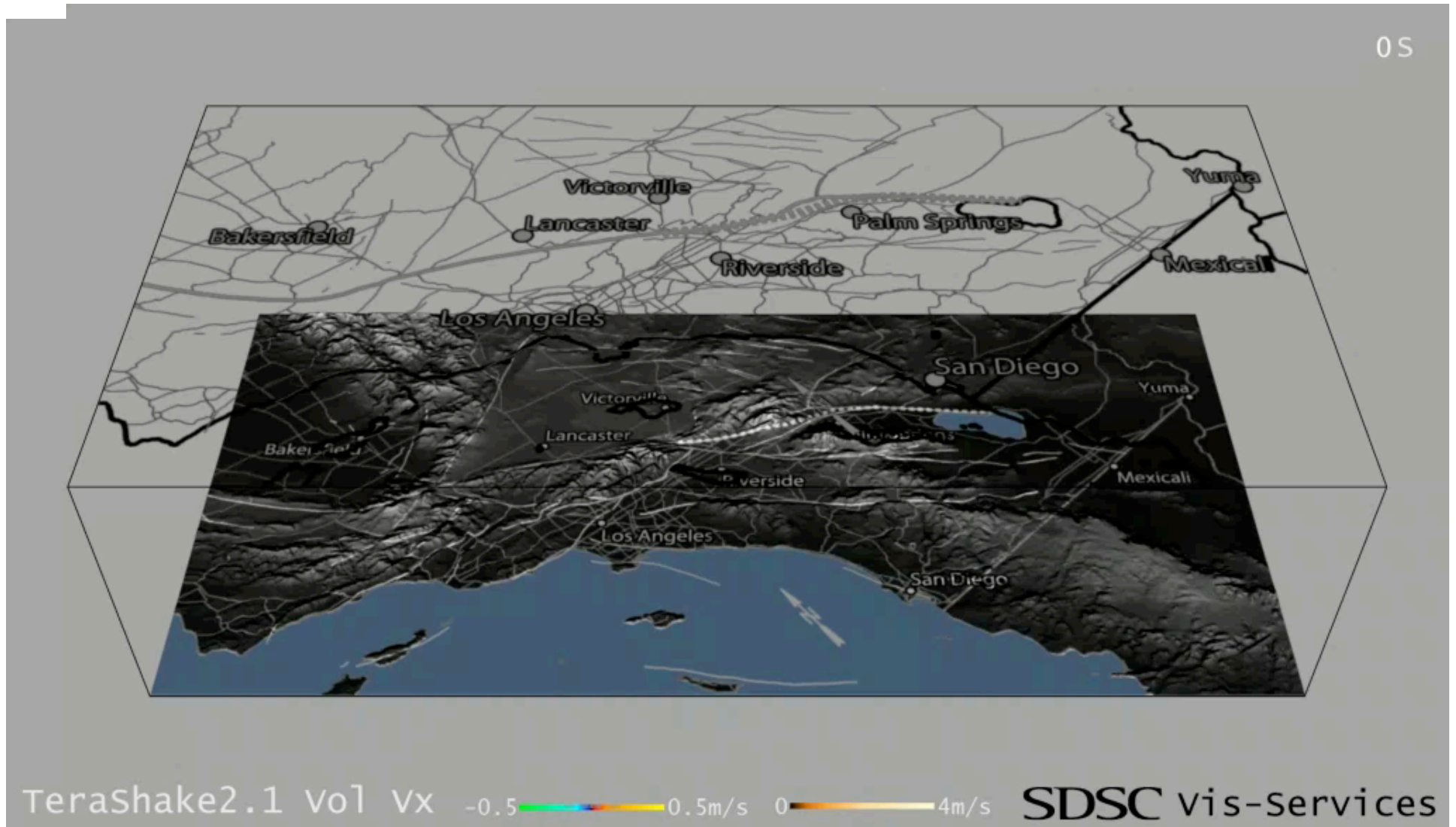
How TeraShake Works

- 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



DataCentral@SDSC

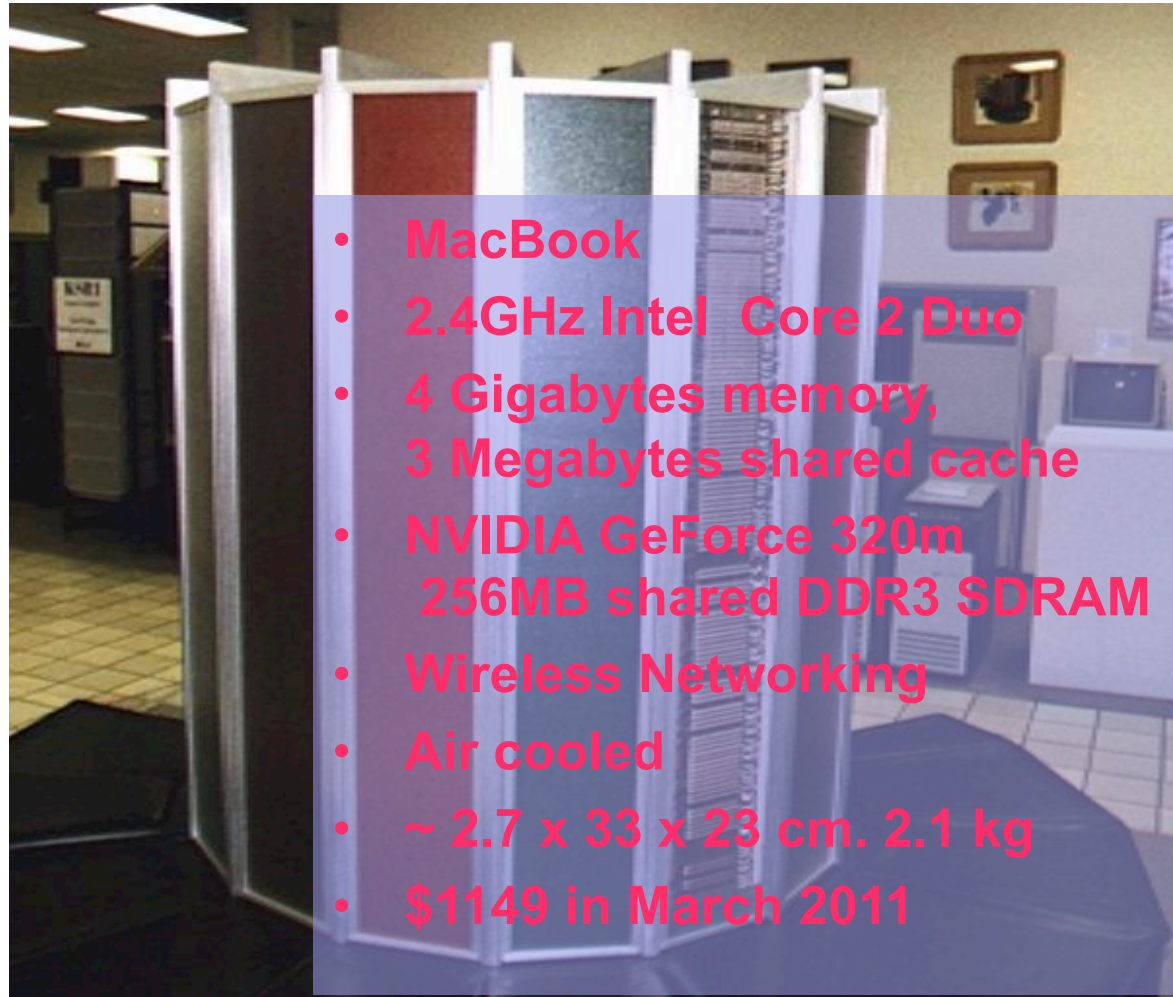
Animation



The advance of technology

Today's laptop would have been yesterday's supercomputer

- Cray-1 Supercomputer
- 80 MHz processor
- 8 Megabytes memory
- Water cooled
- 1.8m H x 2.2m W
- 4 tons
- Over \$10M in 1976



- MacBook
- 2.4GHz Intel Core 2 Duo
- 4 Gigabytes memory, 3 Megabytes shared cache
- NVIDIA GeForce 320m 256MB shared DDR3 SDRAM
- Wireless Networking
- Air cooled
- ~ 2.7 x 33 x 23 cm. 2.1 kg
- \$1149 in March 2011

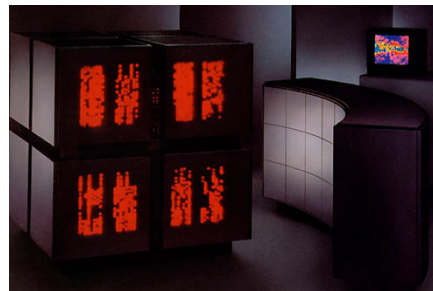
Technological disruption

- **Transformational:** modelling, healthcare...
- Challenges
 - New wisdom for delivering a solution
 - Manage software development costs

**Cray-1, 1976,
240 Megaflops**



**Connection Machine
CM-2, 1987**



**Beowulf cluster,
late 1990s**



**Nvidia Tesla, 4.14
Tflops, 2009**



**Intel 48 core
processor, 2009**



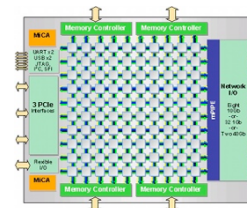
**ASCI Red,
1997, 1Tflop**



**Sony Playstation 3,
150 Gflops, 2006**

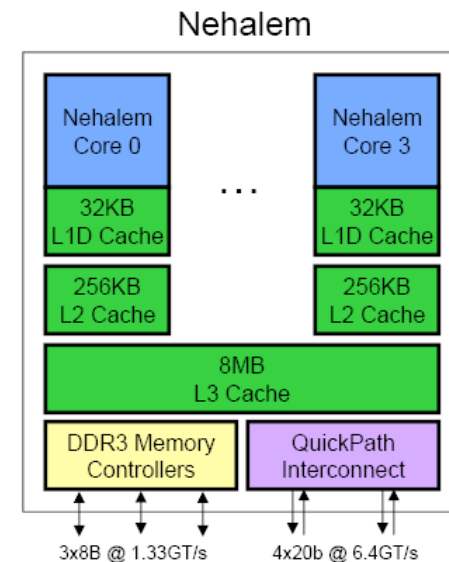
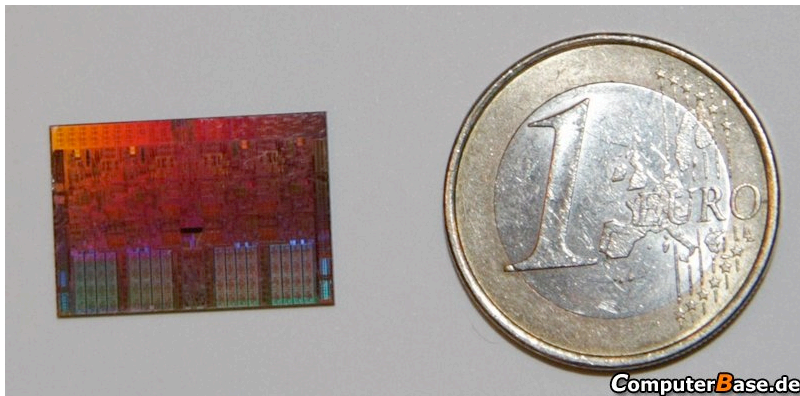


**Tilera 100 core
processor, 2009**



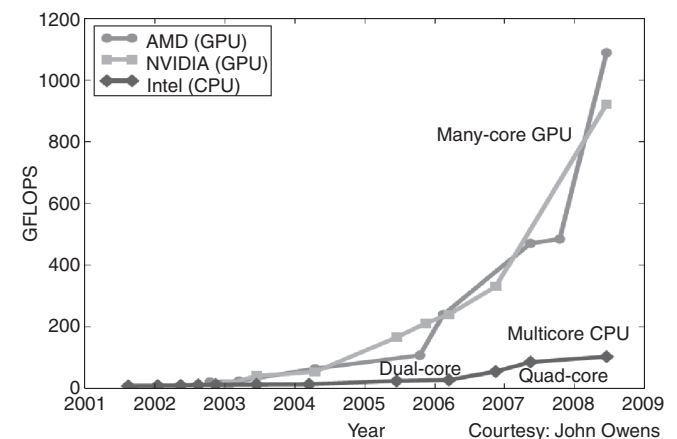
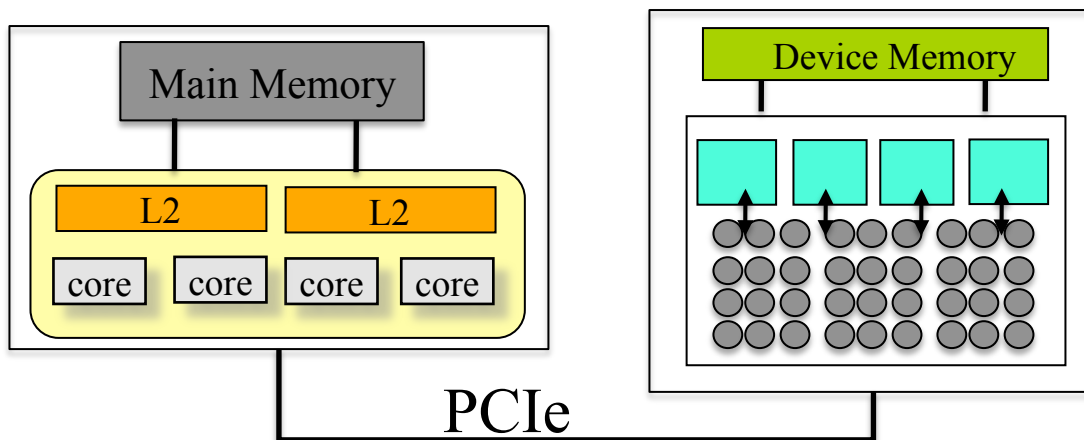
The age of the multi-core processor

- On chip parallel computer
- IBM Power4 (2001), many others follow (Intel, AMD, Tiler, Cell Broadband Engine)
- First dual core laptops (2005-6)
- GPUs (nVidia, ATI): supercomputer on a desktop



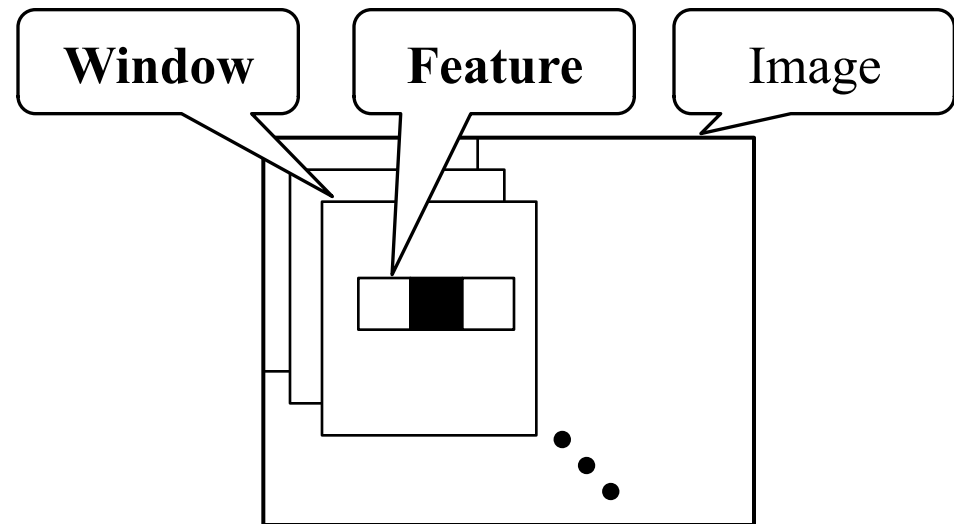
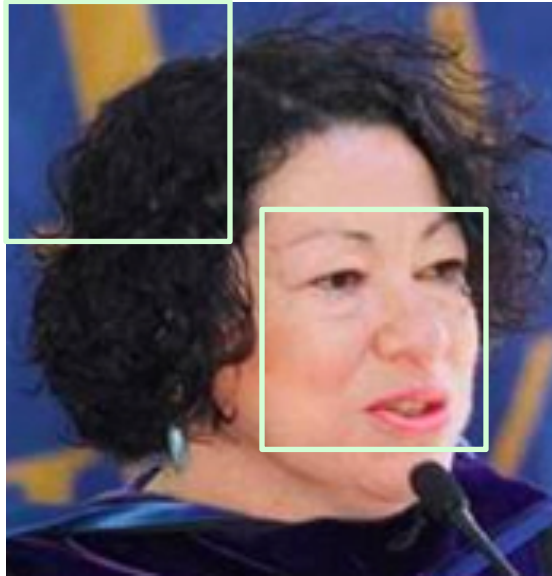
Latest disruption: the NVIDIA GPU family

- Specialized many many core processor
- SIMT execution: piecewise SIMD on long vectors
- Massive virtual multithreading, fine grained
- Explicitly manage the memory hierarchy
- Rapidly changing landscape



Face detection with Viola-Jones algorithm

- Searches images for **features** of a human face



- GPU performance competitive with FPGAs, but far lower development cost

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 justified the effort
 - Enabled new scientific discovery
 - Software development costs were reasonable

How hard is it?

- Two types of users
 - Enjoy the capabilities that parallelism provides w/o being aware of the details, e.g. photoshop
 - Get into the driver's seat: write parallel programs, enjoy the benefits of customization, personal preferences
- A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be reformulated
- There is no magic compiler that can turn a serial program into an efficient parallel program *all the time* and on *all machines*

What is involved?

- Parallelism introduces many new tradeoffs
 - Redesign the software
 - Rethink the problem solving technique
- Performance programming
 - Low-level details: heavily application dependent
 - Irregularity in the computation and its data structures forces us to think even harder
- Techniques and tools that help us

Memory hierarchies

Address space organization

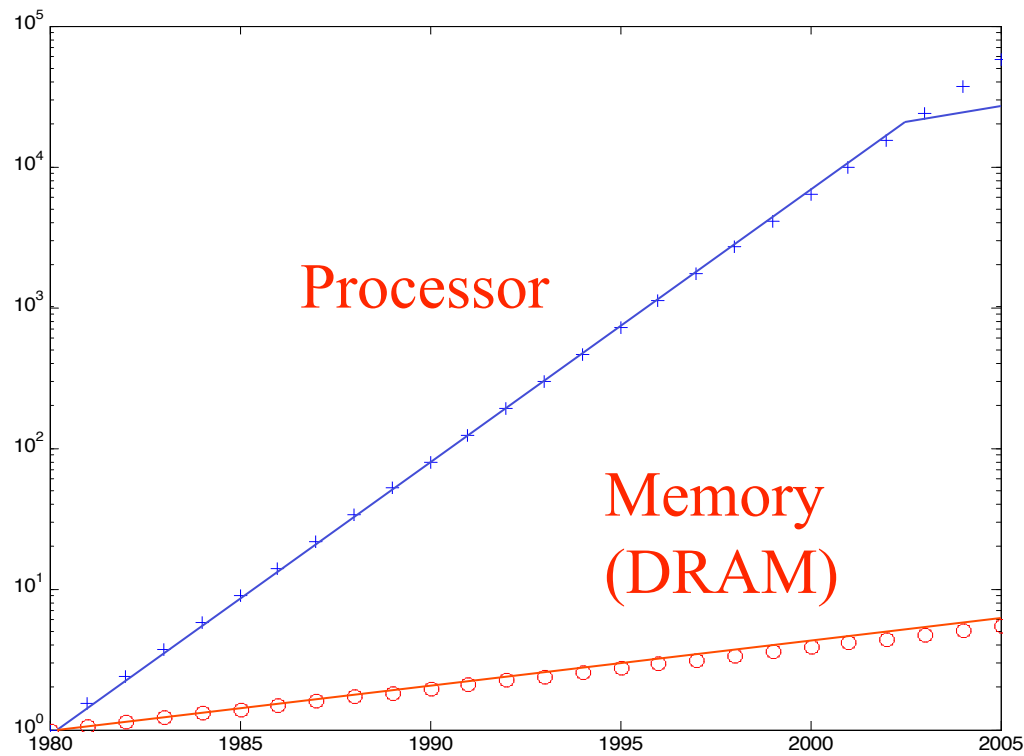
Control

The hardware

- Address space organization
 - Shared memory
 - Distributed memory
- Control mechanism

The processor-memory gap

- The result of technological trends
- Difference in processing and memory speeds growing exponentially over time



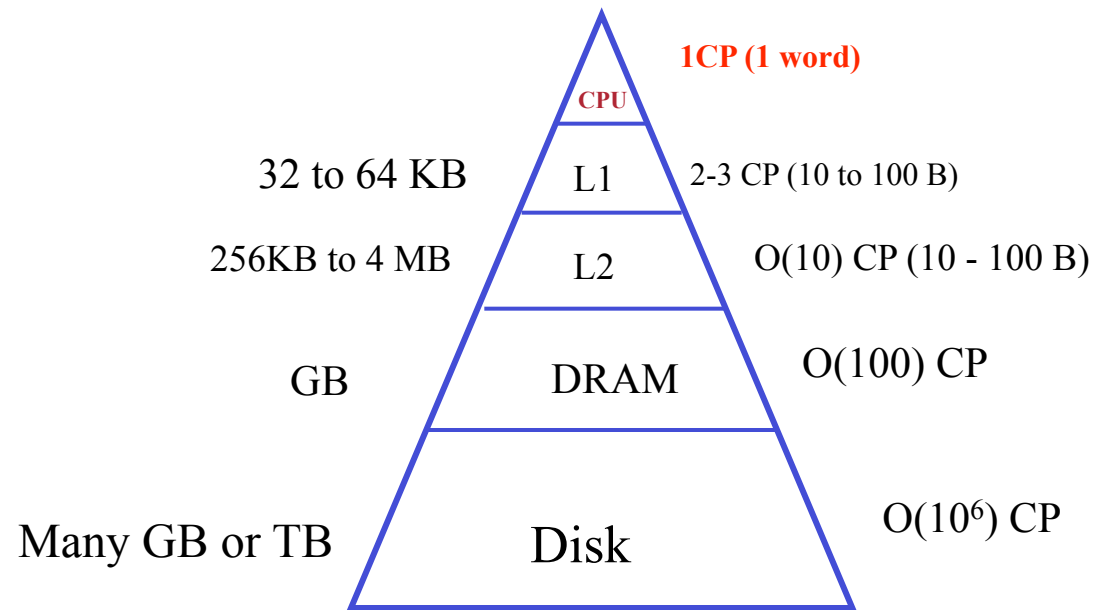
An important principle: locality

- Programs generally exhibit two forms of locality in accessing memory
 - Temporal locality (time)
 - Spatial locality (space)
- Often involves loops
- Opportunities for reuse

```
for t=0 to T-1
  for i = 1 to N-2
     $u[i] = (u[i-1] + u[i+1]) / 2$ 
```

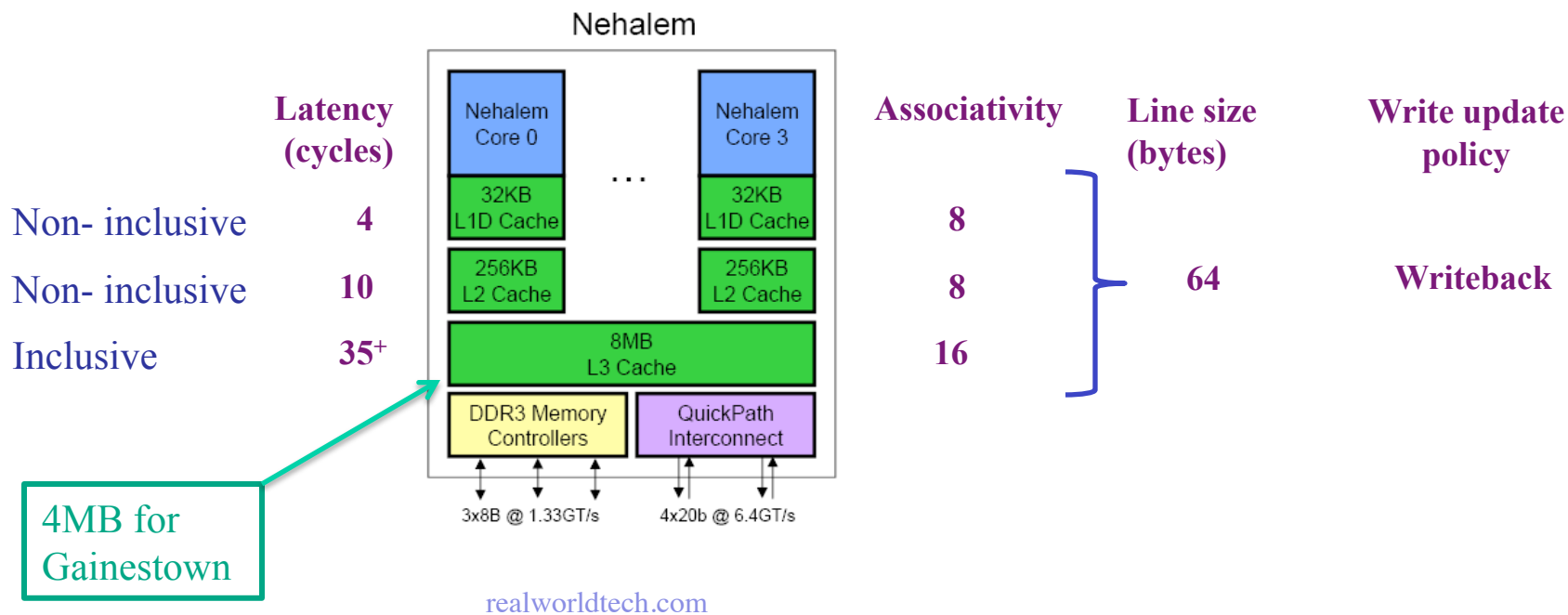
Memory hierarchies

- Exploit reuse through a hierarchy of smaller but faster memories
- Put things in faster memory if we reuse them frequently



Nehalem's Memory Hierarchy

- Source: *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Table 2.7

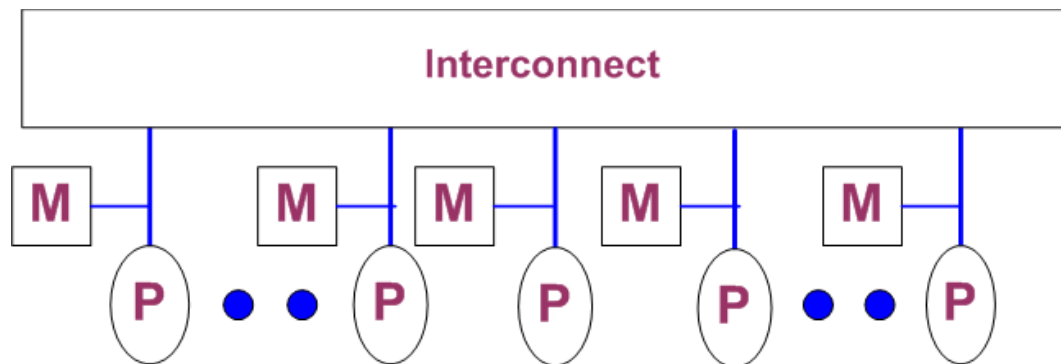
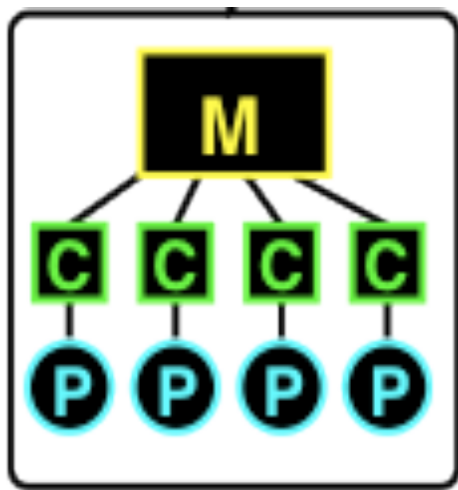


Address Space Organization

- We classify the address space organization of a parallel computer according to whether or not it provides global memory
- If there is global memory we have a “shared memory” or “shared address space” architecture
 - *multiprocessor* vs *partitioned global address space*
- When there is no global memory, we have a “shared nothing” architecture, also known as a *multicomputer*

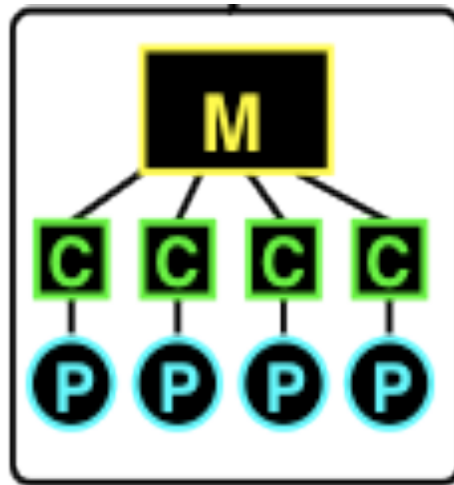
Multiprocessor organization

- Hardware automatically performs the global to local mapping using address translation mechanisms
- 2 types, according to uniformity of memory access times
 - **UMA**: Uniform Memory Access time
 - **NUMA**: Non-Uniform Memory Access time



UMA shared memory

- Uniform Memory Access time
- In the absence of contention, all processors observe the same memory access time
- Also called **Symmetric Multiprocessors**
- Usually bus based
- Not scalable



Intel Clovertown Memory Hierarchy

- Ieng-203
- Intel Xeon X5355 (Intro: 2006)
- Two “Woodcrest” dies on a multichip module



techreport.com/articles.x/10021/2

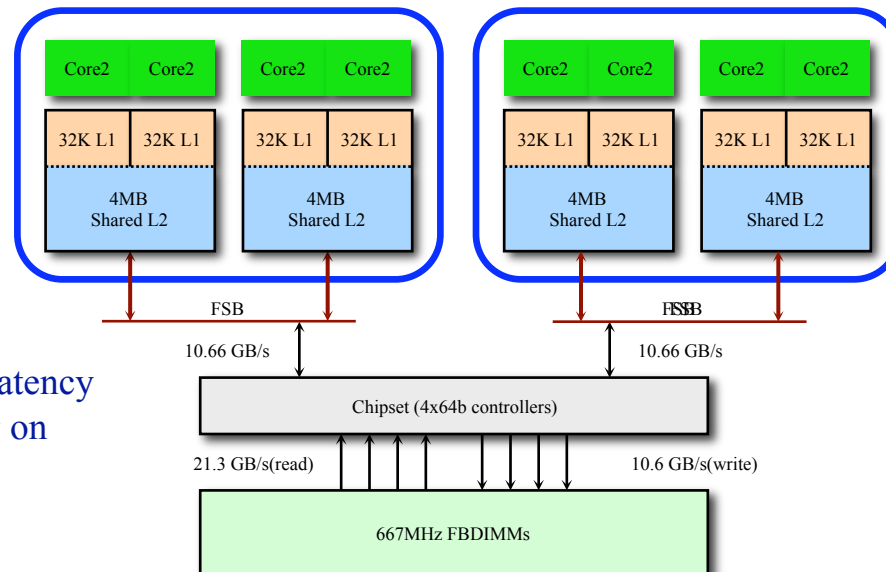
Line Size = 64B (L1 and L2)

Access latency (clocks)

3

14*

* Software-visible latency will vary depending on access patterns and other factors



Associativity

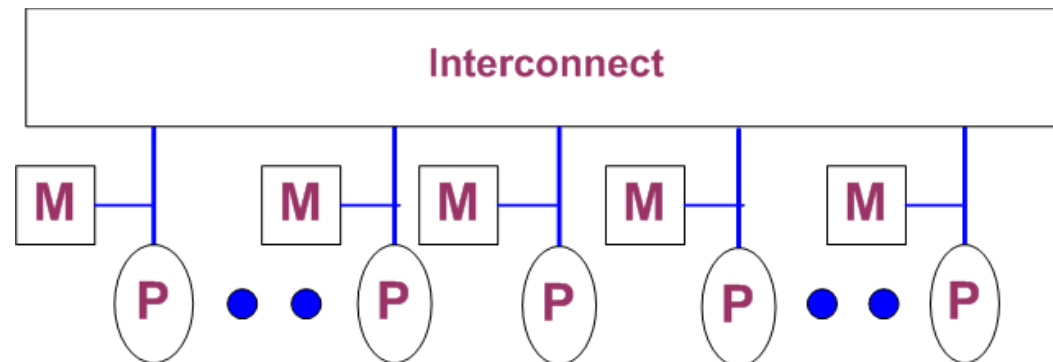
8

16

Sam Williams et al.

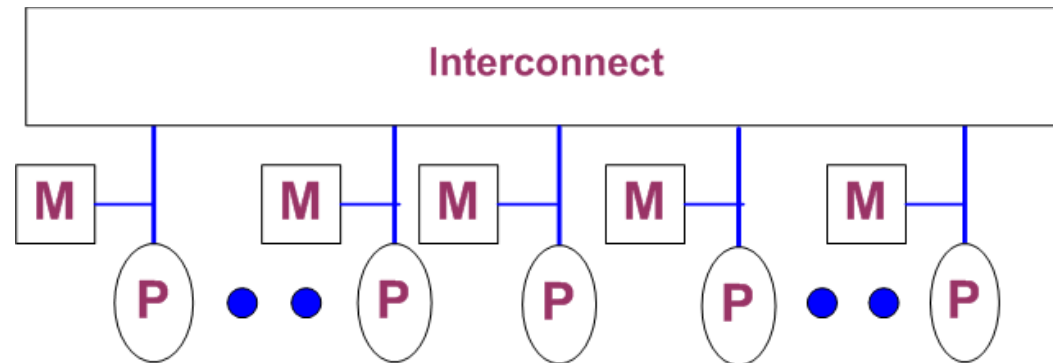
NUMA

- 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 Altix UV, up to 1024 cores
 - Dash prototype at San Diego Supercomputer Center
 - Software/hardware support to monitor sharers



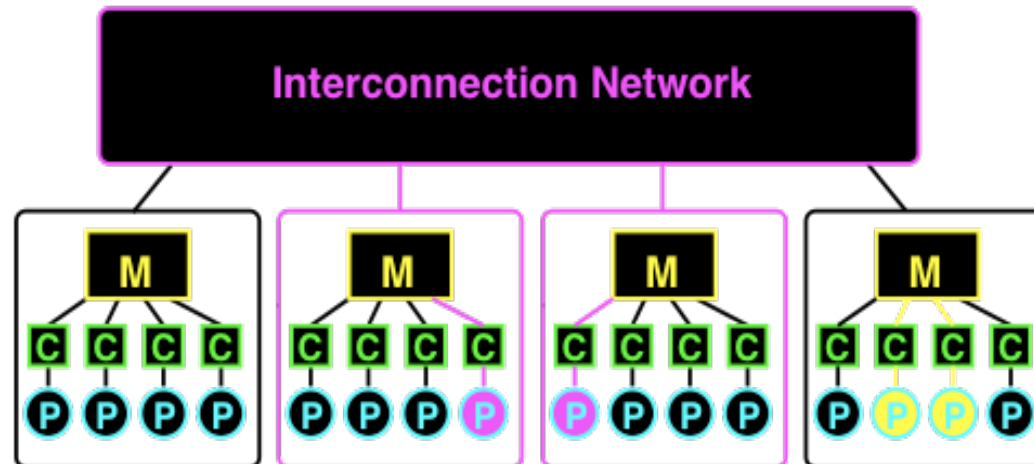
Architectures without shared memory

- A 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*



Hybrid organizations

- Multi-tier organizations are hierarchically organized
- Each node is a multiprocessor, usually an SMP
- Nodes communicate by passing messages, processors within a node communicate via shared memory
- All clusters and high end systems today



Parallel processing this course

- Hardware
 - Mainframe
 - GPUs
- Primary programming models
 - MPI
 - CUDA
- Alternatives
 - Threads
 - Non-traditional (actors, dataflow)

The hardware

- Address space organization
 - Shared memory
 - Distributed memory
- Control mechanism

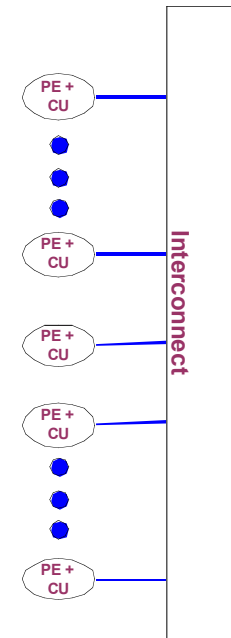
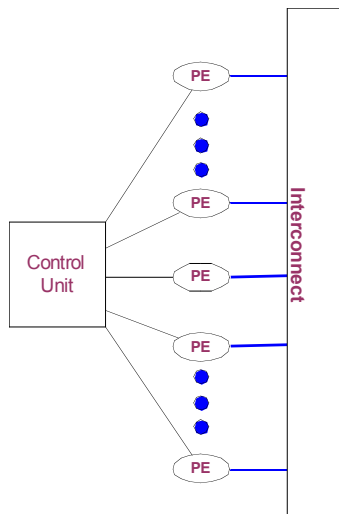
Control Mechanism

Flynn's classification (1966)

How do the processors issue instructions?

SIMD: Single Instruction, Multiple Data

Execute a global instruction stream in lock-step



MIMD: Multiple Instruction, Multiple Data
Clusters and servers processors execute
instruction streams independently

SIMD (Single Instruction Multiple Data)

- Operate on regular arrays of data
- Two landmark SIMD designs
 - ILIAC IV (1960s)
 - Connection Machine 1 and 2 (1980s)
- Vector computer: Cray-1 (1976)
- Intel and others support SIMD for multimedia and graphics
 - SSE
 - Streaming SIMD extensions, AltiVec
 - Operations defined on vectors
- GPUs, Cell Broadband Engine
- Reduced performance on data dependent or irregular computations

$$\begin{bmatrix} 2 \\ 4 \\ 8 \\ 7 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 5 \\ 2 \end{bmatrix}$$

```
forall i = 0 : n-1  
    x[i] = y[i] + z [ K[i] ]  
end forall
```

```
forall i = 0 : n-1  
    if ( x[i] < 0) then  
        y[i] = x[i]  
    else  
        y[i] = √x[i]  
    end if  
end forall
```

Covered in today's lecture

- Motivation for parallel processing
- Technological disruption
- Programming issues
- Hardware organization and technology

Fin