

CSE 260 – Introduction to Parallel Computation

Topic 6: Models of Parallel Computers

October 11-18, 2001

Models of Computation

What's a model good for??

- Provides a way to think about computers.

Influences design of:

- Architectures
 - Languages
 - Algorithms
- Provides a way of estimating how well a program will perform.

Cost in model should be roughly same as cost of executing program

Outline

- RAM model of sequential computing
- PRAM
- Fat tree
- PMH
- BSP
- LogP

The Random Access Machine Model

RAM model of serial computers:

- Memory is a sequence of words, each capable of containing an integer.
- Each memory access takes one unit of time
- Basic operations (add, multiply, compare) take one unit time.
- Instructions are not modifiable
- Read-only input tape, write-only output tape

Has RAM influenced our thinking?

Language design:

No way to designate registers, cache, DRAM.

Most convenient disk access is as streams.

How do you express atomic read/modify/write?

Machine & system design:

It's not very easy to modify code.

Systems pretend instructions are executed in-order.

Performance Analysis:

Primary measures are operations/sec (MFlop/sec, MHz, ...)

What's the difference between Quicksort and Heapsort??

What about parallel computers

- RAM model is generally considered a very successful “bridging model” between programmer and hardware.
- “Since RAM is so successful, let’s generalize it for parallel computers ...”

PRAM [Parallel Random Access Machine]

(Introduced by Fortune and Wyllie, 1978)

PRAM composed of:

- P processors, each with its own unmodifiable program.
- A single shared memory composed of a sequence of words, each capable of containing an arbitrary integer.
- a read-only input tape.
- a write-only output tape.

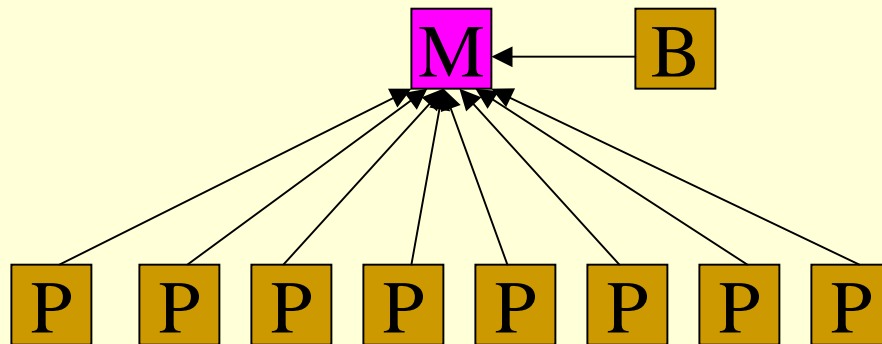
PRAM model is a synchronous, MIMD, shared address space parallel computer.

More PRAM taxonomy

- Different protocols can be used for reading and writing shared memory.
 - EREW - exclusive read, exclusive write
A program isn't allowed to have two processors access the same memory location at the same time.
 - CREW - concurrent read, exclusive write
 - CRCW - concurrent read, concurrent write
Needs protocol for arbitrating write conflicts
 - CROW - concurrent read, owner write
Each memory location has an official "owner"
- PRAM can emulate a message-passing machine by partitioning memory into private memories.

Broadcasting on a PRAM

- “Broadcast” can be done on CREW PRAM in $O(1)$ steps:
 - Broadcaster sends value to shared memory
 - Processors read from shared memory



- Requires $\lg(P)$ steps on EREW PRAM.

Finding Max on a CRCW PRAM

- We can find the max of N distinct numbers $x[1], \dots, x[N]$ in **constant time** using N^2 procs!

Number the processors P_{rs} with $r, s \in \{1, \dots, N\}$.

- Initialization: P_{1s} sets $A[s] = 1$.
- Eliminate non-max's: if $x[r] < x[s]$, P_{rs} sets $A[r] = 0$.
Requires concurrent reads & writes.
- Find winner: If $A[r] = 1$, P_{r1} sets $\text{max} = x[r]$.

Some questions

1. What if the $x[i]$'s aren't necessarily distinct?
2. Can you sort N numbers in constant time?
And only use only N^k processors (for some k)?
3. How fast can you sort on CREW?
4. Does any of this have any practical significance ?????

PRAM is **not** a great success

- Many theoretical papers about fine-grained algorithmic techniques and distinctions between various modes.
- Results seem irrelevant.
 - Performance predictions are inaccurate.
 - Hasn't lead to programming languages.
 - Hardware doesn't have fine-grained synchronous steps.

Fat Tree Model

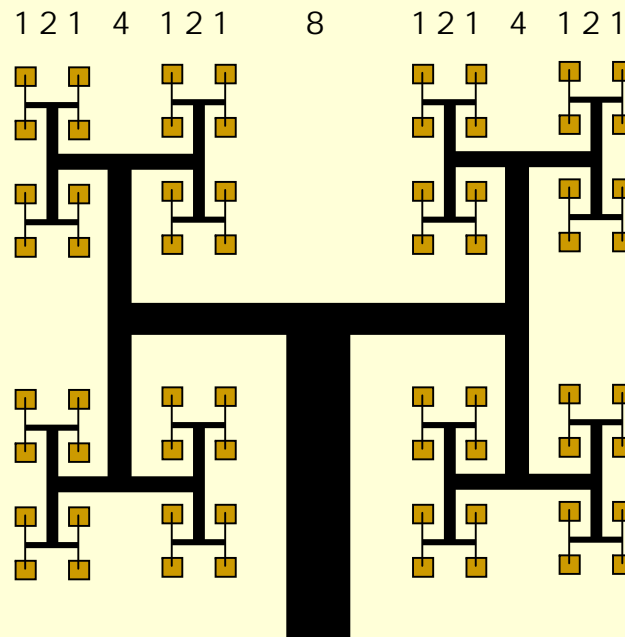
(Leiserson, 1985)

Processors at leaves of tree

Group of k^2 processors connected by k -width bus

k^2 processors fit in $(k \lg 2k)^2$ area

Area-universal: can simulate t steps of any p -proc computer in $t \lg p$ steps.



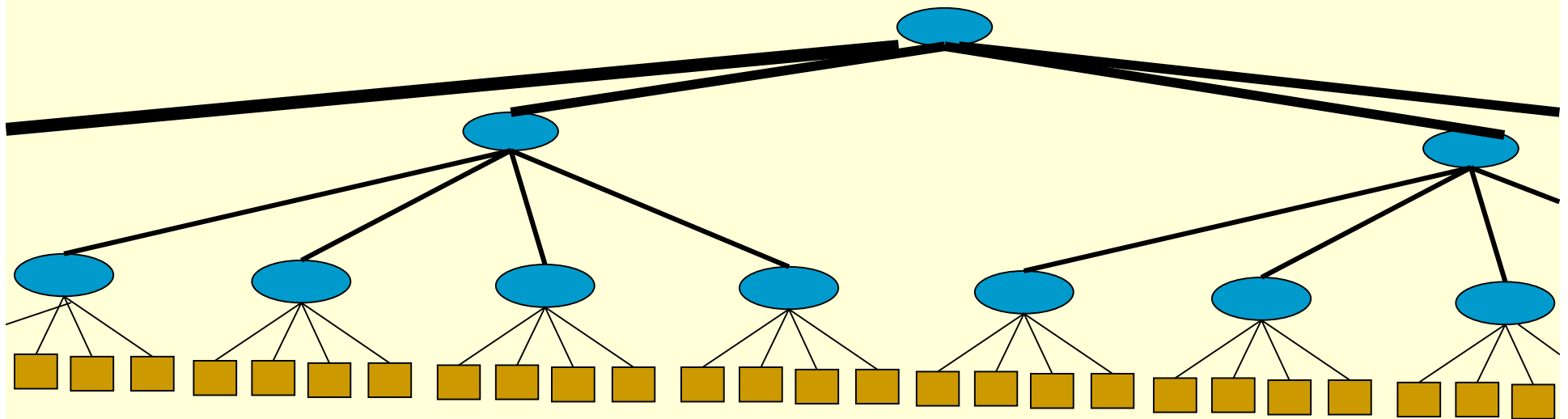
Fat Tree Model inspired CM-5

Up to 1024 nodes in fat tree

- 20MB/sec/node within group-of-4
- 10MB/sec/node within group-of-16
- 5 MB/sec/node among larger groups

Node = 33MHz Sparc plus 4 33 MFlop/sec vector units

Plus fast narrow "control network" for parallel prefix operations



What happened to fat trees?

- CM-5 had many interesting features
 - Active message VSM software layer.
 - Randomized routing.
 - Fast control network.
- It was somewhat successful, but died anyway
 - Using the floating point unit well wasn't easy.
 - Perhaps not sufficiently COTS-like to compete.
- Fat trees live on, but aren't highlighted ...
 - IBM SP and others have less bandwidth between cabinets than within a cabinet.
 - Seen more as a flaw than a feature.

Another look at the RAM model

- RAM analysis says matrix multiply is $O(N^3)$.

```
for i = 1 to N
```

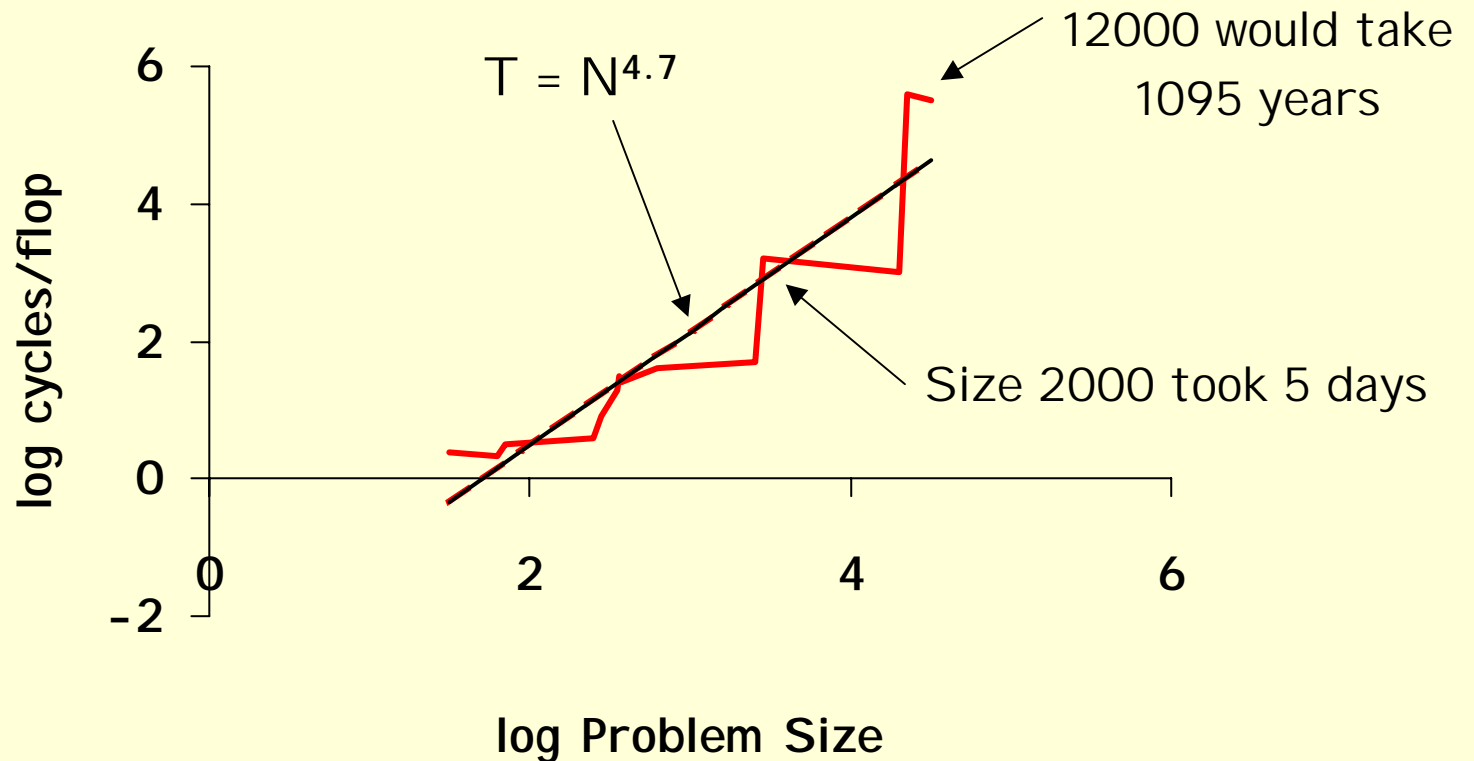
```
  for j = 1 to N
```

```
    for k = 1 to N
```

```
      C[i,j] += A[i,k]*B[k,j]
```

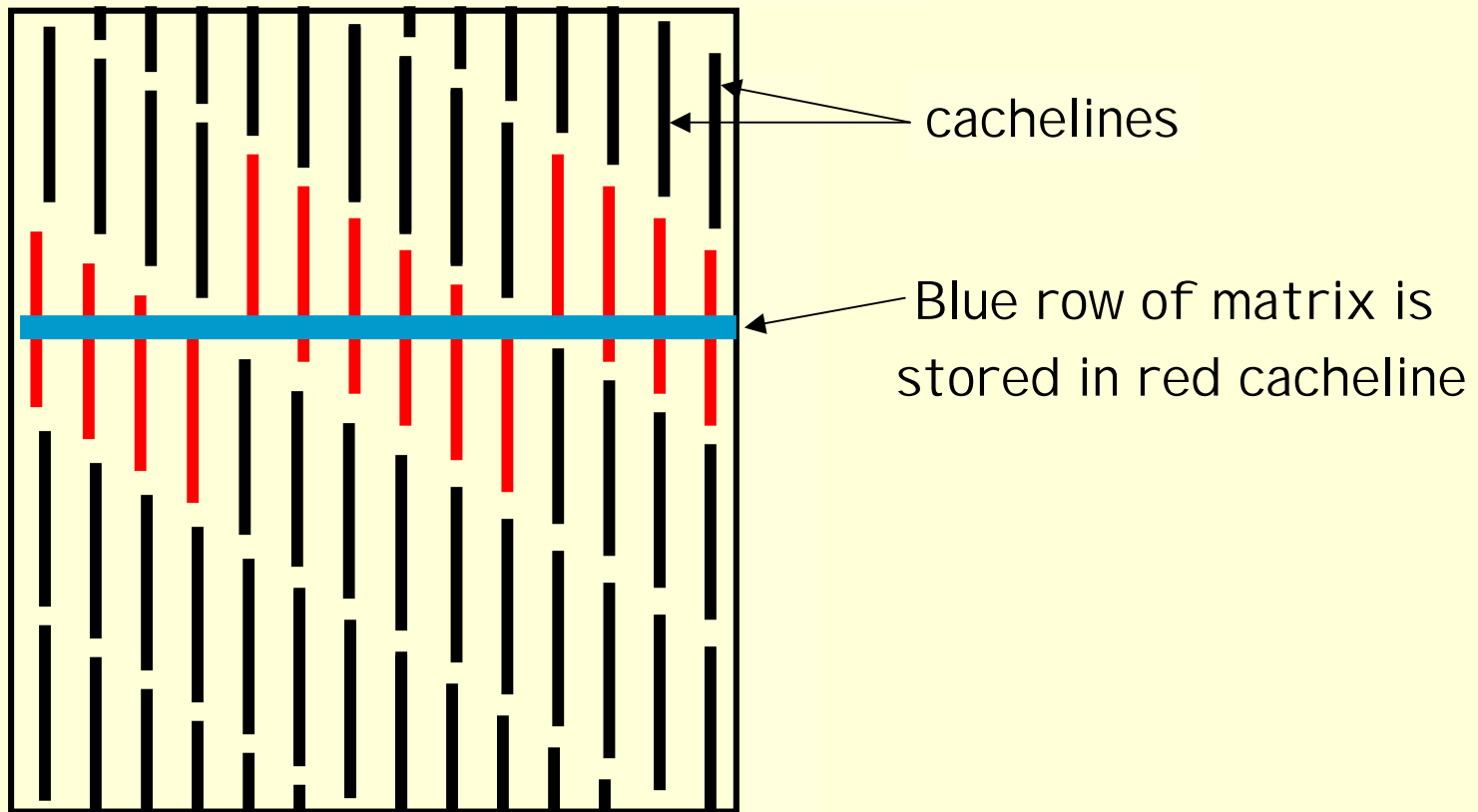
- Is it??

Matrix Multiply on RS/6000



$O(N^3)$ performance would have constant cycles/flop
Performance looks much closer to $O(N^5)$

Column major storage layout



Memory Accesses in Matrix Multiply

```
for i = 1 to N
```

```
  for j = 1 to N
```

```
    for k = 1 to N
```

```
      C[i,j] += A[i,k]*B[k,j]
```

Stride-N
access to
one row*

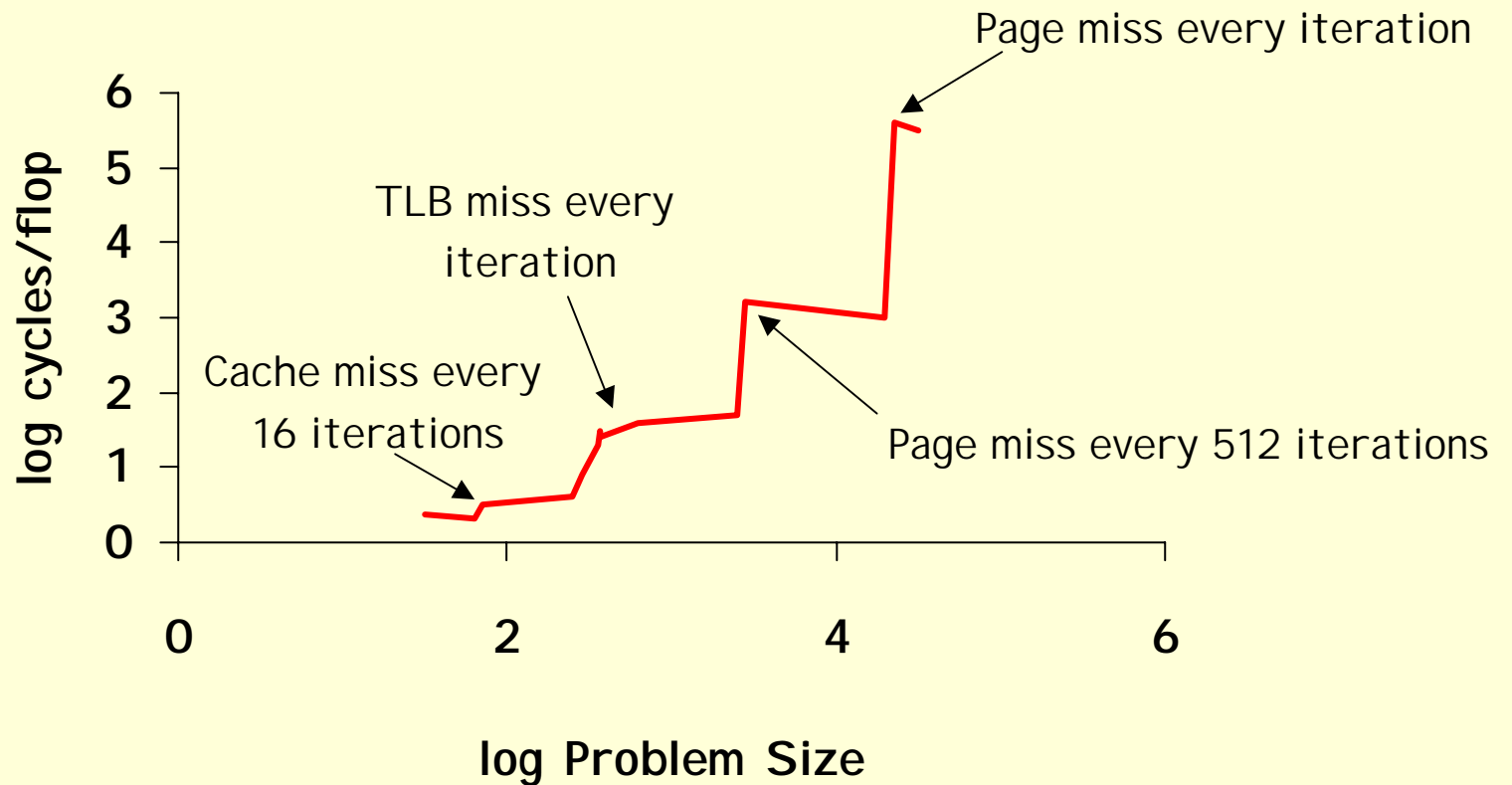
Sequential
access through
entire matrix

When cache (or TLB or memory) can't hold entire B matrix, there will be a miss on every line.

When cache (or TLB or memory) can't hold a row of A, there will be a miss on each access

* assumes data is in column-major order

Matrix Multiply on RS/6000



Where are we?

- RAM model says naive matrix multiply is $O(N^3)$
- Experiments show it's $O(N^5)$ -ish
- Explanation involves cache, TLB, and main memory limits and block sizes
- Conclusion: memory features are important and should be included in model.

Models of memory behavior

Uniprocessor models looking at data access costs:

Two-level models (main memory & cache):

Floyd ('72), Hong & Kung ('81)

Hierarchical Memory Model

Accessing memory location i costs $f(i)$

Aggarwal, Alpern, Chandra & Snir ('87)

Block Transfer Model

Moving block of length k at location i costs $k+f(i)$

Aggarwal, Chandra & Snir ('87)

Memory Hierarchy Model

Multilevel memory, block moves, extends to parallelism

Alpern & Carter ('90)

Memory Hierarchy model

A uniprocessor is

Sequence of memory modules

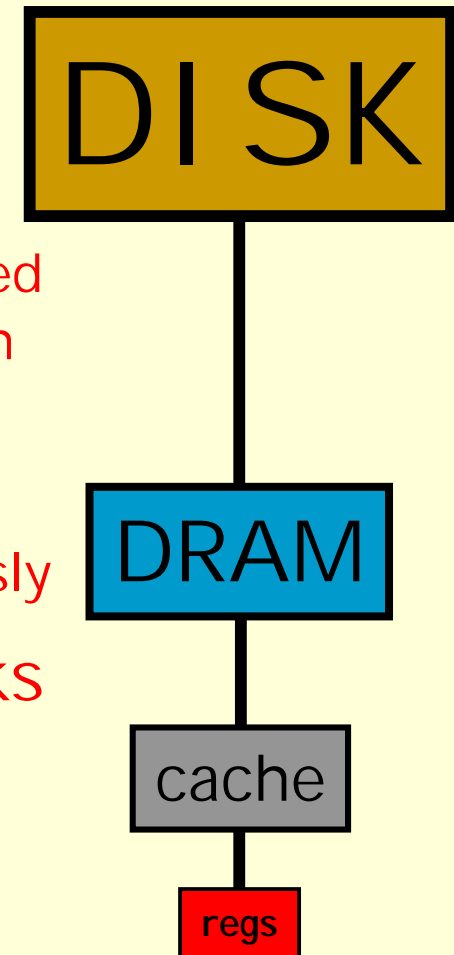
Highest level is large memory, low speed
Processor (level 0) is tiny memory, high speed

Connected by channels

All channels can be active simultaneously

Data are moved in fixed-sized blocks

A block is a chunk of contiguous data
Block size depends on level



Does MH model influence *your* thinking?

Say your computer is a sequence of modules:

You want to move data to the fast one at bottom.

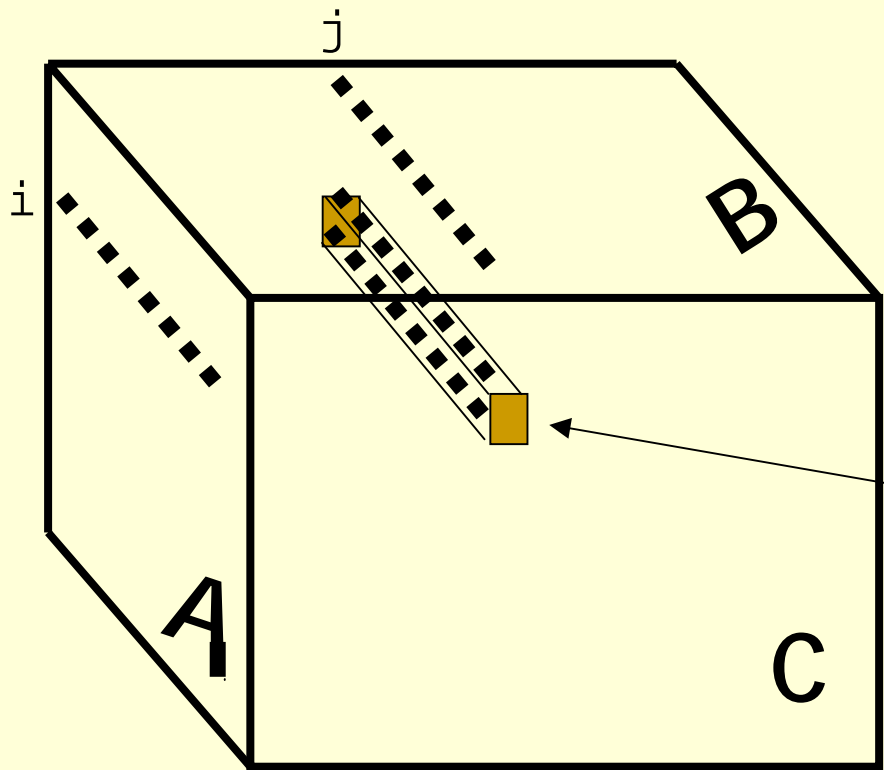
Moving contiguous chunks of data is faster.

How do you accomplish this??

One possible answer: divide & conquer

(Mini project – does the model suggest anything for your favorite algorithm?)

Visualizing Matrix Multiplication

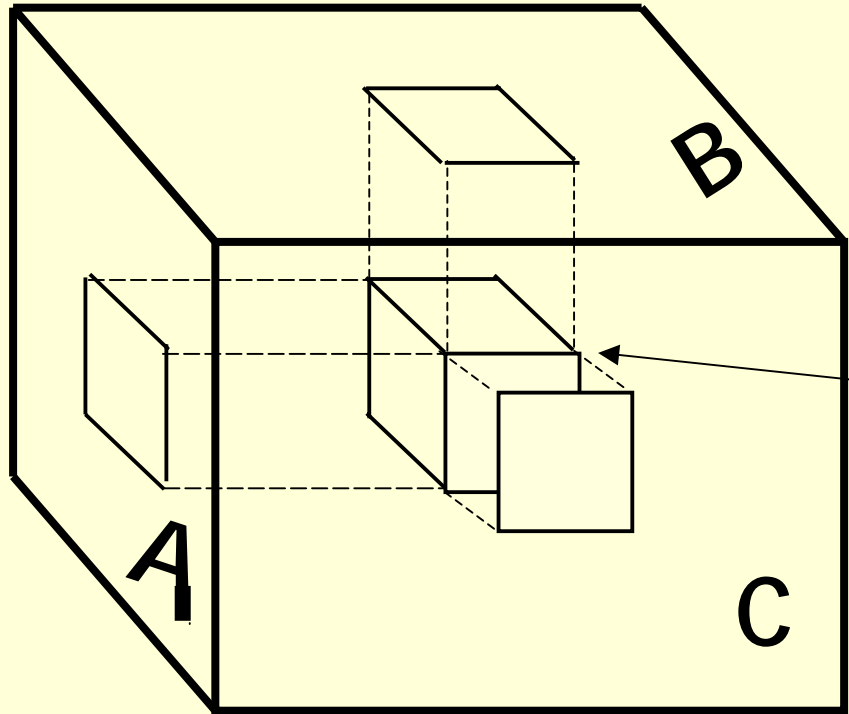


$$C = A B$$

"stick" of computation
is dot product of a
row of A with column of B

$$c_{ij} = \sum a_{ik} * b_{kj}$$

Visualizing Matrix Multiplication



"Cubelet" of computation is product of a submatrix of A with submatrix of B

- Data involved is proportional to surface area.
- Computation is proportional to volume.

MH algorithm for $C = AB$

Partition computation into “cubelets”

Each cubelet requires $s \times s$ submatrix of A and B

$3 s^2$ data needed; allows s^3 multiply-adds

Parent module gives child sequence of cubelets.

Choose s to ensure all data fits into child's memory

Child sub-partitions cubelet into still smaller pieces.

Known as “blocking” or “tiling” long before MH model invented (but rarely applied recursively).

Theory of MH algorithm for $C = AB$

“Uniform” Memory Hierarchy (UMH) model looks similar to actual computers.

Block size, number of blocks per module, and transfer time per item grow by constant factor per level.

Naive matrix multiplication is $O(N^5)$ on UMH.

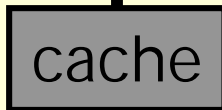
Similar to observed performance.

Tiled algorithm is $O(N^3)$ on UMH.

Tiled algorithm gets about 90% “peak performance” on many computers.

Moral: good MH algorithm \leftrightarrow good in practice.

Visualizing computers in MH model



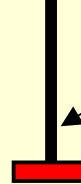
This computer is reasonably well-balanced

29

Height of module = $\lg(\text{blocksize})$

Width = $\lg(\text{number of blocks})$

Length of channel = $\lg(\text{transfer time})$



Doesn't satisfy "wide cache principle" (square submatrices don't fit).

Bandwidth too low

This one isn't

Parallel Memory Hierarchy (PMH) model

Alpern & Carter: "Since MH model is so great, let's generalize it for parallel computers!"

A computer is a *tree* of memory modules

Largest memory is at root.

Children have less memory, more compute power.

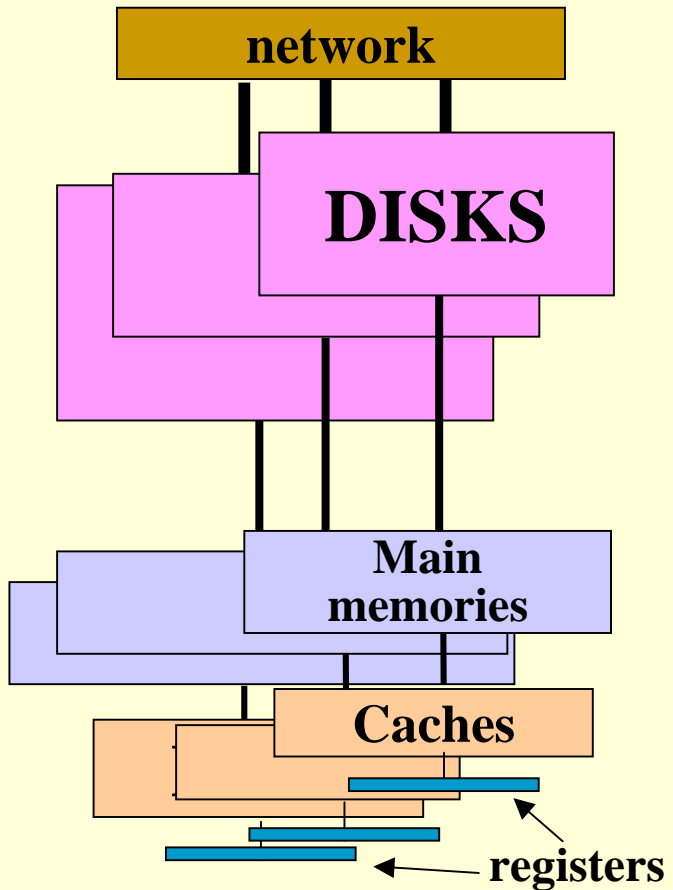
Four parameters per module

Block size, number of blocks, transfer time from parent, and number of children.

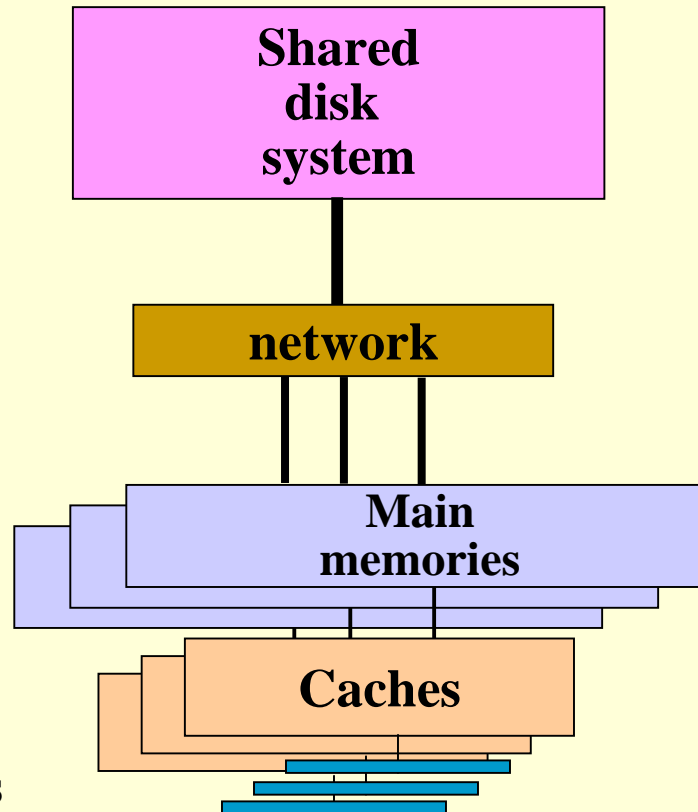
Homogeneous \leftrightarrow all modules at a level have same parameters

(PMH ignores difference between shared and distributed address space computation.)

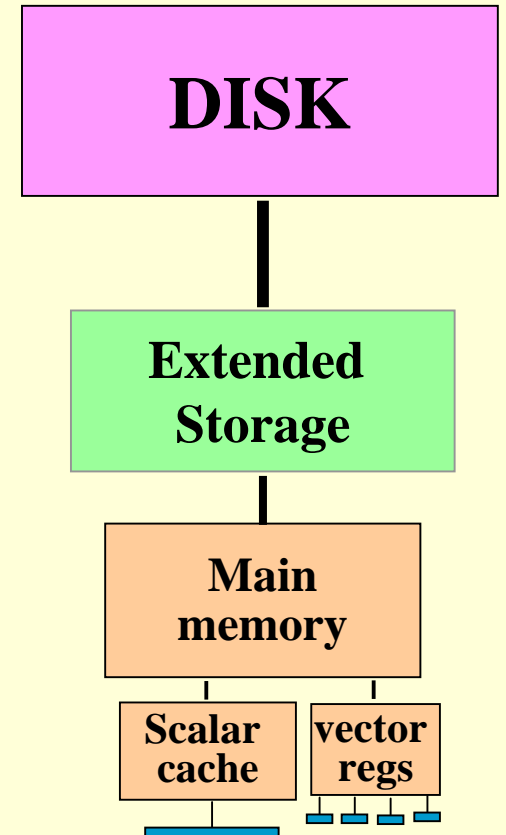
Some Parallel Architectures



The Grid

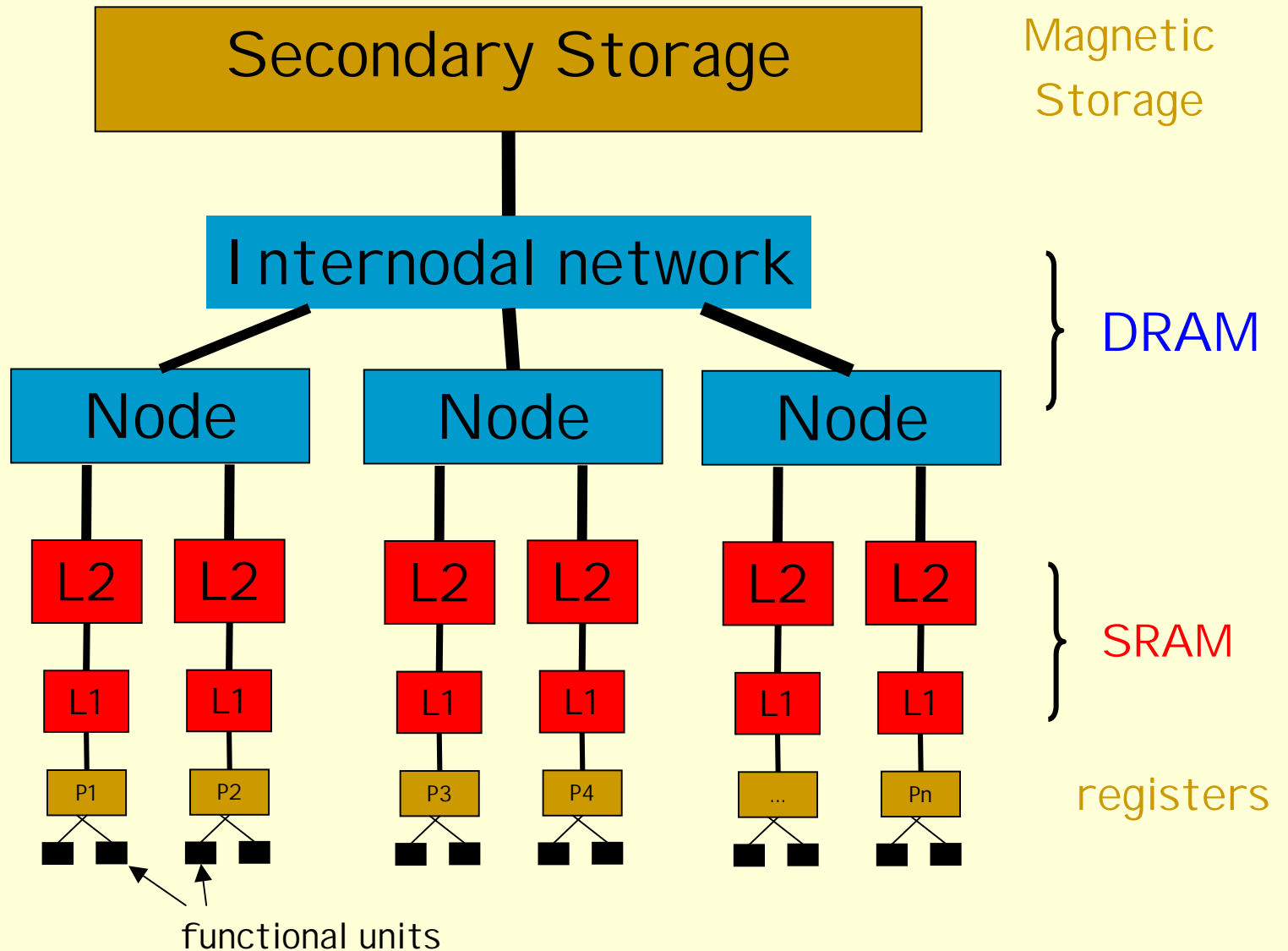


NOW



Vector supercomputer

PMH model of multi-tier computer



Observations

- PMH can model heterogeneous systems as well as homogeneous ones.
- More expensive computers have more parallelism and higher bandwidth near leaves
- Computers getting more levels & more branching.
- Parallelizing code for PMH is very similar to tuning it for a memory hierarchy.
 - Break computation into *independent* blocks
 - Send blocks of work to children

Needed for
parallelization



BSP (Bulk Synchronous Parallel) Model

Valiant, "A Bridging Model for Parallel Computation", CACM, Aug '90

CORRECTION!!

I have been confusing BSP with the "Phase PRAM" model (Gibbons, SPAA '89), which indeed is a shared-memory model with periodic barrier synchronizations.

In BSP, each processor has local memory.

"One-sided" communication style is advocated.

- There are globally-known "symbolic addresses" (like VSM)
 - Data may be inconsistent until next barrier synchronization
 - Valiant suggests hashing implementation of puts and gets.

BSP Programs

- BSP programs composed of **supersteps**.
- In each superstep, processors execute up to L computational steps using locally stored data, and also can send and receive messages
- Processors synchronize at end of superstep (at which time all messages have been received)
- Oxford BSP is a library of C routines for implementing BSP programs. It provides:
 - **Direct Remote Memory Access (a VSM layer)**
 - **Bulk Synchronous Message Passing (sort of like non-blocking message passing in MPI)**

superstep

synch

superstep

synch

superstep

synch

Parameters of BSP Model

P = number of processors.

s = processor speed (steps/second).

observed, not "peak".

L = time to do a barrier synchronization (steps/synch).

g = cost of sending message (steps/word).

measure g when all processors are communicating.

h_0 = minimum # of messages per superstep.

For $h \geq h_0$, cost of sending h messages is hg .

h_0 is similar to block size in PMH model.

BSP Notes

- Number of processors in model can be greater than number of processors of machine.

Easier for computer to complete the remote memory operations

- Not all processors need to join barrier synch
- Time for superstep = $1/s \times$
(max (operations performed by any processor)
+ $g \times$ max (messages sent or received by a
processor, h_0)
+ L)

Some representative BSP parameters

Machine (all have $P=8$)	MFlop/s s	Flops/synch L	Flops/word g	words (32b) $n_{1/2}$ for h_0
Pentium II NOW switched Ethernet	88	18300	31	32
Cray T3E	47	506	1.2	40
IBM SP2	26	5400	9	6
Pentium NOW serial Ethernet 1	61	540,000	2800	61

From oldwww.comlab.ox.ac.uk/oucl/groups/bsp/index.html (1998)

NOTE: Benchmarks for determining s were not tuned.

LogP Model

- Developed by Culler, Karp, Patterson, etc.
 - Famous guys at Berkeley
- Models communication costs in a multicomputer.
- Influenced by MPP architectures (circa 1993), notably the CM-5.
 - each node is a powerful processor with large memory
 - interconnection structure has limited bandwidth
 - interconnection structure has significant latency

LogP parameters

- **L**: latency - time for message to go from P_{sender} to P_{receiver}
- **o**: overhead - time either processor is occupied sending or receiving message
 - Processor can't do anything else for o cycles.
- **g**: gap - minimum time between messages
 - Processor can have at most $\lceil L/g \rceil$ messages in transit at a time.
 - Gap includes overhead time (so $\text{overhead} \leq \text{gap}$)
- **P**: number of processors

L, o, and g are measured in cycles

Efficient Broadcasting in LogP

Picture shows $P=8$, $L=6$, $g=4$, $o=2$

