

Part II: Architecture

Goal: Understand the main properties of parallel computers

The parallel approach to computing ... does require that some original thinking be done about numerical analysis and data management in order to secure efficient use. In an environment which has represented the absence of the need to think as the highest virtue, this is a decided disadvantage.

-- Dan Slotnick, 1967

What's The Deal With Hardware?

☐ Facts Concerning Hardware

- Parallel computers differ dramatically from each other -- there is no standard architecture
 - ☐ No single programming target!
- Parallelism introduces costs not present in vN machines -- communication; influence of external events
- Many parallel architectures have failed
- Details of parallel computer ~~are~~ ^{should be} of no greater concern to programmers than details of vN

The “no single target” is key problem to solve

Our Plan

- Think about the problem abstractly
 - Introduce instances of basic || designs
 - Multicore
 - Symmetric Multiprocessors (SMPs)
 - Large scale parallel machines
 - Clusters
 - Blue Gene/L
 - Formulate a model of computation
 - Assess the model of computation
-

Shared Memory

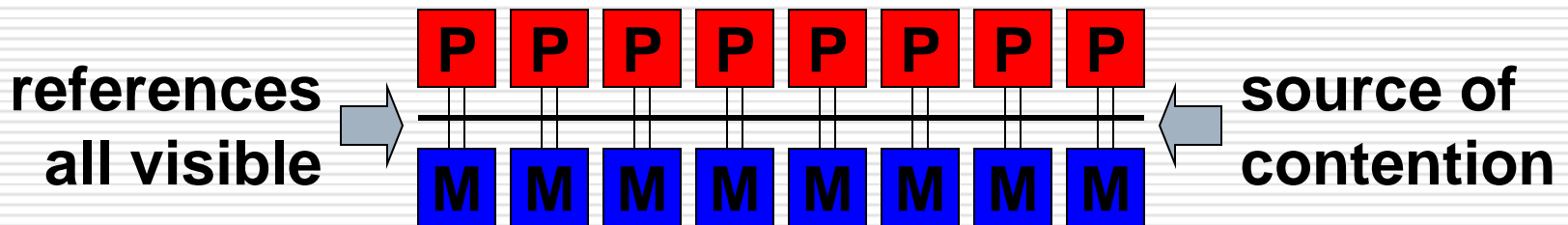
- Global memory shared among \parallel processors is the natural generalization of the sequential memory model
 - Thinking about it, programmers *assume* sequential consistency when they think \parallel ism
 - Recall Lamport's definition of SC:
 - "...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."
-

Sequential Consistency

- ❑ SC difficult to achieve under all circumstances
 - ❑ [Whether SC suffices as a model at all is a deep and complex issue; there's more to say than today's points.]
 - ❑ The original way to achieve SC was literally to keep a single memory image and make sure that modifications are recorded in that memory
-

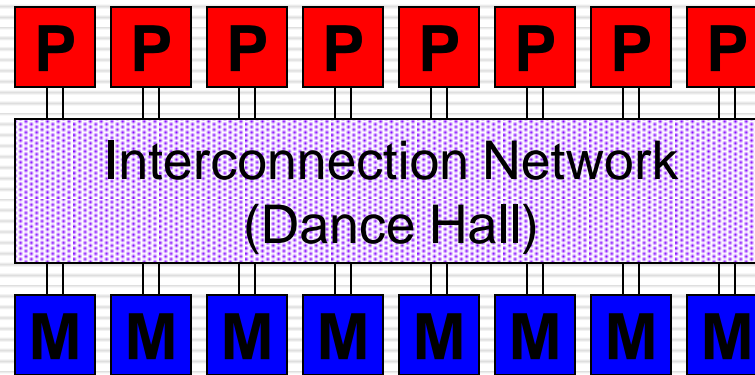
The Problem

- The “single memory” view implies ...
 - The memory is the **only source** of values
 - Processors use memory values **one-at-a-time**, not sharing or caching; if not available, stall
 - Lock when fetched, Execute, Store & unlock
- A bus can do this, but ...




Reduce Contention

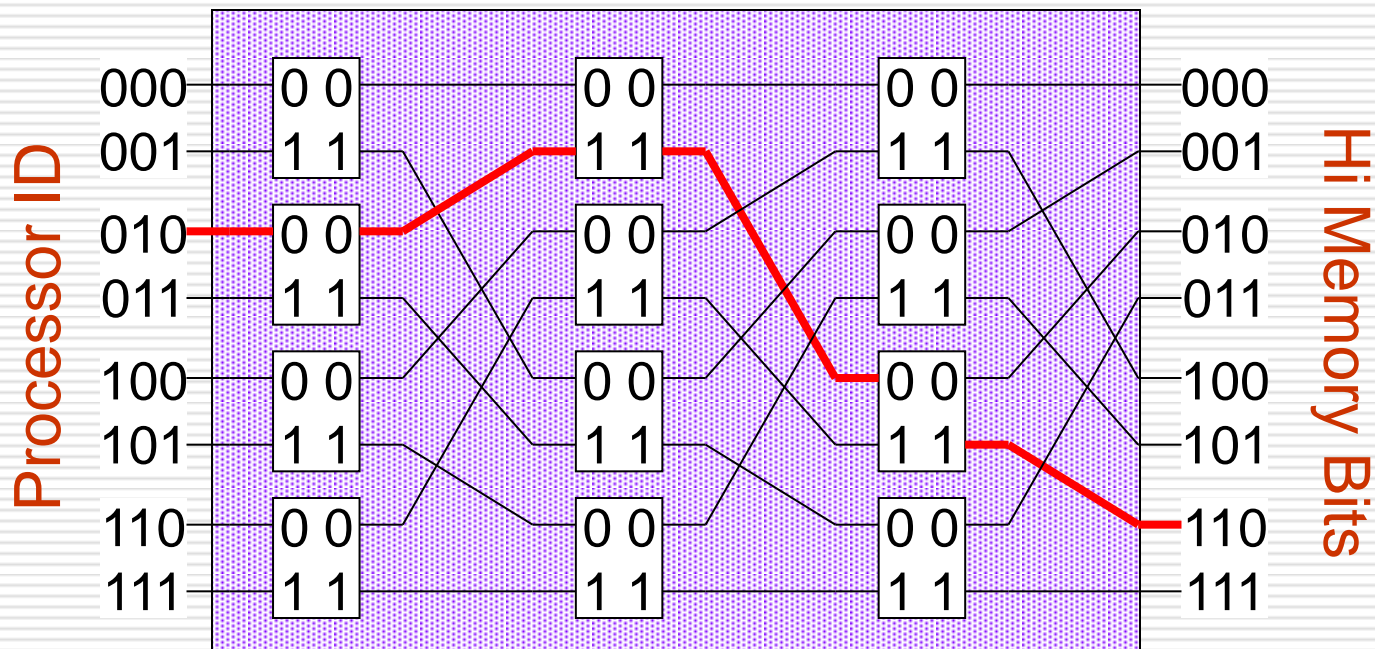
- ❑ Replace bus with network, an early design



- ❑ Network delays cause memory latency to be higher for a single reference than with a the bus, but simultaneous use should help when many references are in the air (MT)
-

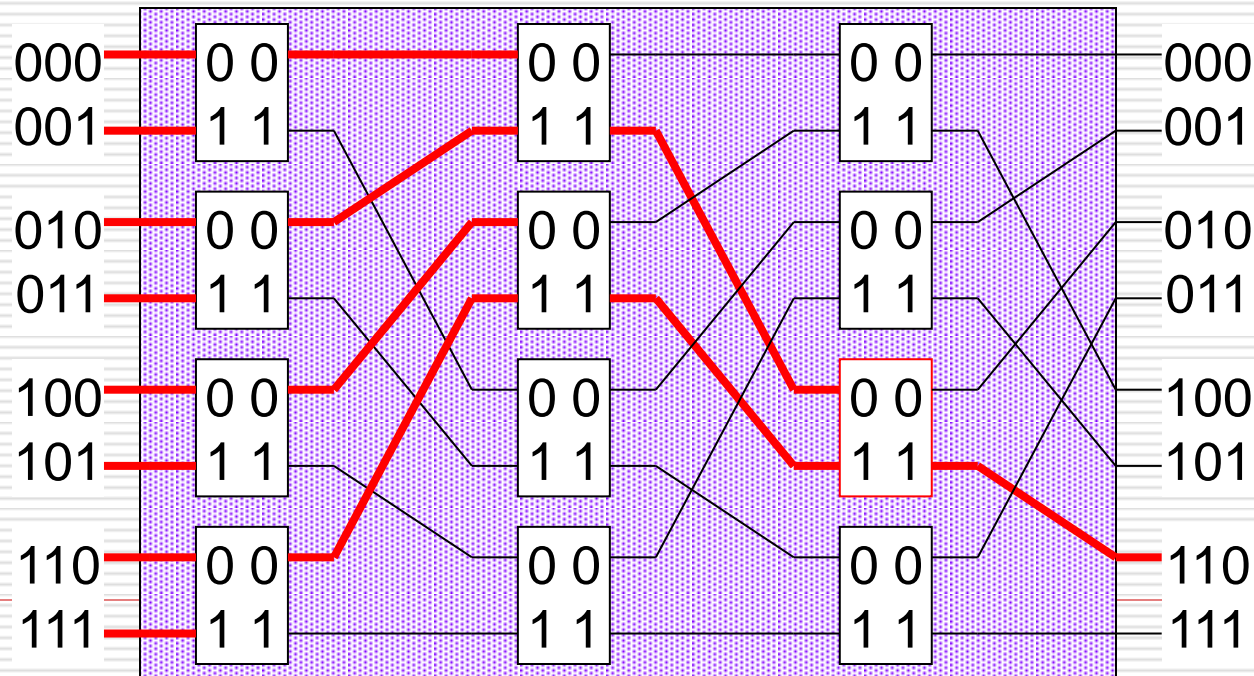
An Implementation

- ❑ Ω -Network is one possible interconnect
- ❑ Processor 2 references memory 6 (110) 



Backing Up In Network

- ❑ Even if processors work on different data, the requests can back up in the network
- ❑ Everyone references data in memory 6



One-At-A-Time Use

- ❑ The critical problem is that only one processor at a time can use/change data
 - Cache read-only data (& pgms) only
 - Check-in/Check-out model most appropriate
 - Conclusion: Processors stall a lot ...
- ❑ Solution: Multi-threading
 - When stalled, change to another waiting activity
 - ❑ Must make transition quickly, keeping context
 - ❑ Need ample supply of waiting activities
 - ❑ Available at different granularities

Briefly recall, Multithreading

- ❑ Multithreading: Executing multiple threads “at once”
 - ❑ The threads are, of course, simply sequential programs executing a von Neumann model of computation
 - ❑ Executed “at once” means that the context switching among them is not implemented by the OS, but takes place opportunistically in the hardware ... 3 related cases
-

Facts of Instruction Execution

- ❑ The von Neumann model requires that each instruction be executed to completion before starting the next
 - Once that was the way it worked
 - Now it is a conceptual model
- ❑ Multi-issue architectures start many instructions at a time, and do them when their operands are available leading to **out of order execution**

```
ld r1,0(r2)
add r1,r5
mult r8,r6
sw r1,0(r2)
li r1,0xabc
sw r1,4(r2)
```

Fine Grain Multithreading: Tera

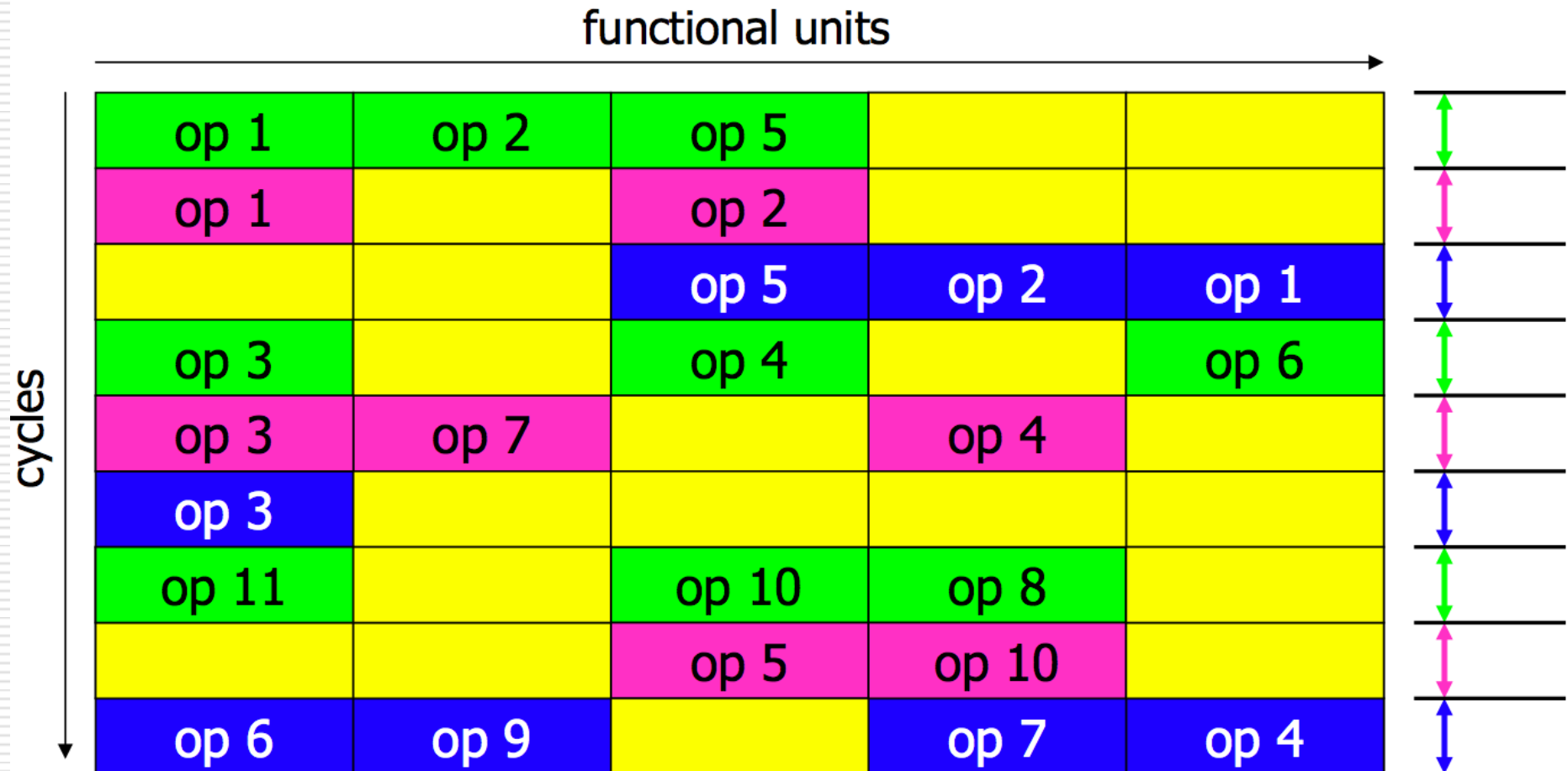
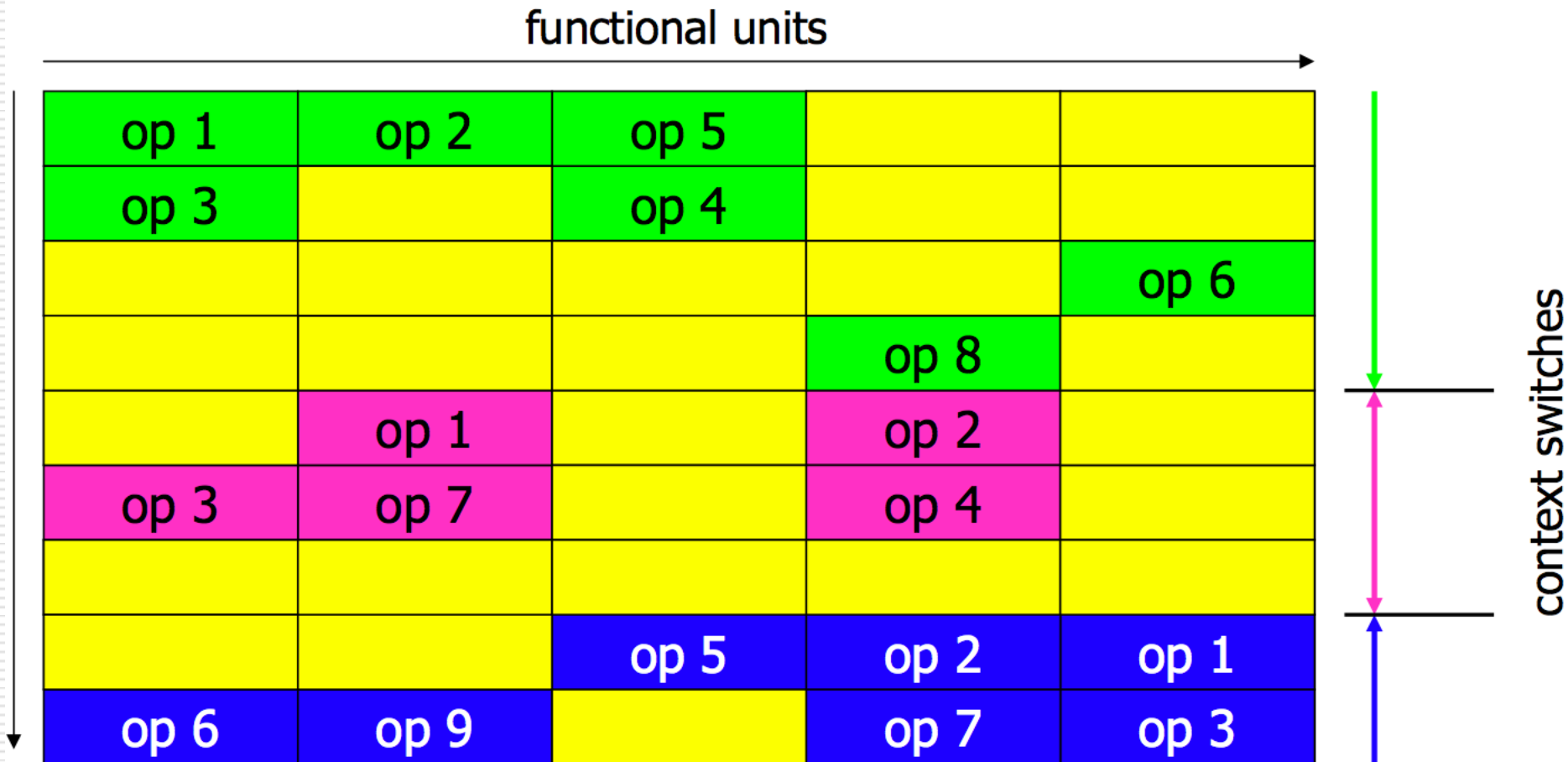
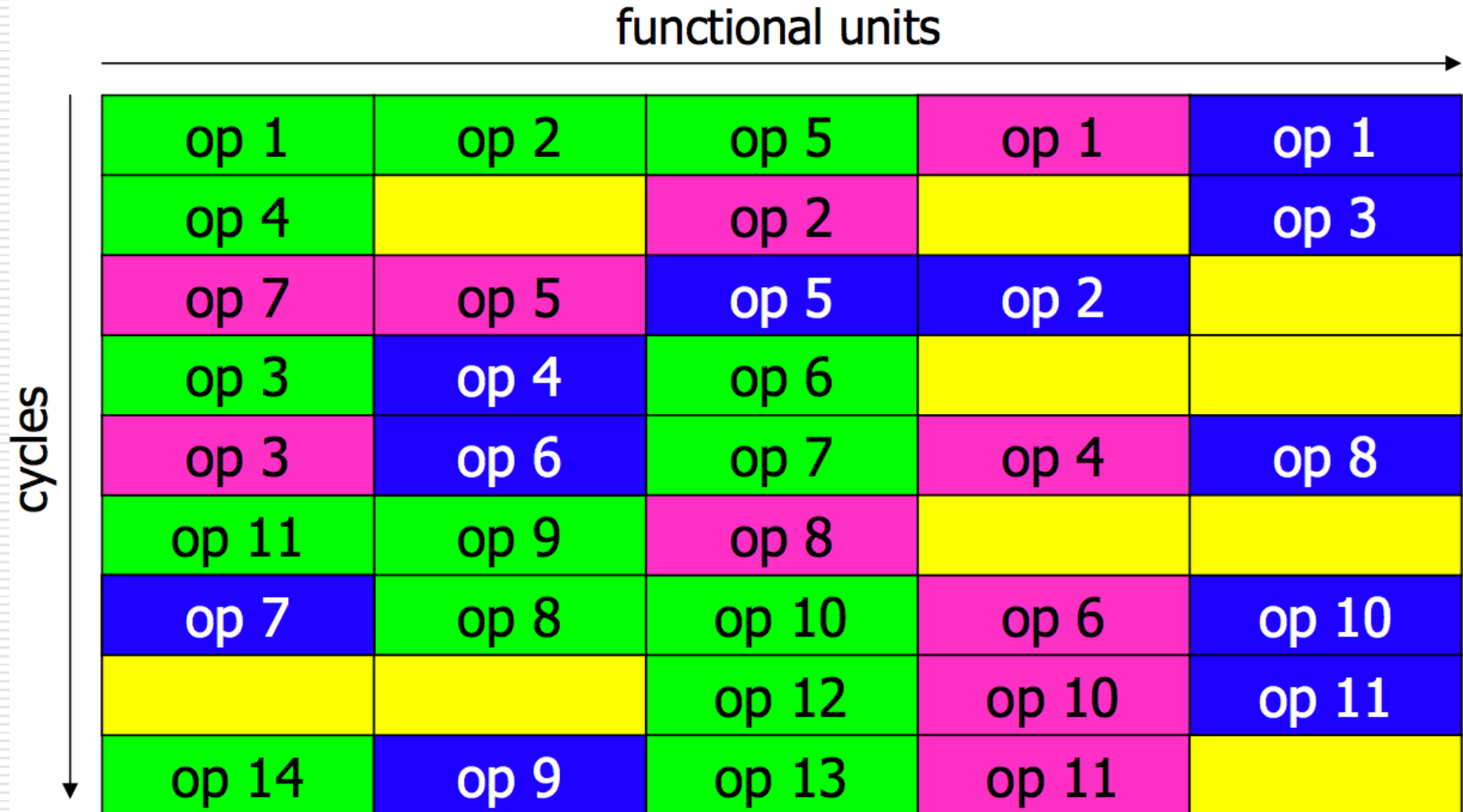


Figure from: Paolo.Ienne@epfl.ch

Coarse Grain Multithreading: Alewife



Simultaneous Multi-threading: SMT



Multi-threading Grain Size

- The point when the activity switches can be
 - Instruction level, at memory reference: Tera MTA
 - Basic block level, with L1 cache miss: Alewife
 - ...
 - At process level, with page fault: Time sharing
 - Another variation (3-address code level) is to execute many threads ($P \cdot \log P$) in batches, called Bulk Synchronous Programming
- **No individual activity improved, but less wait time**

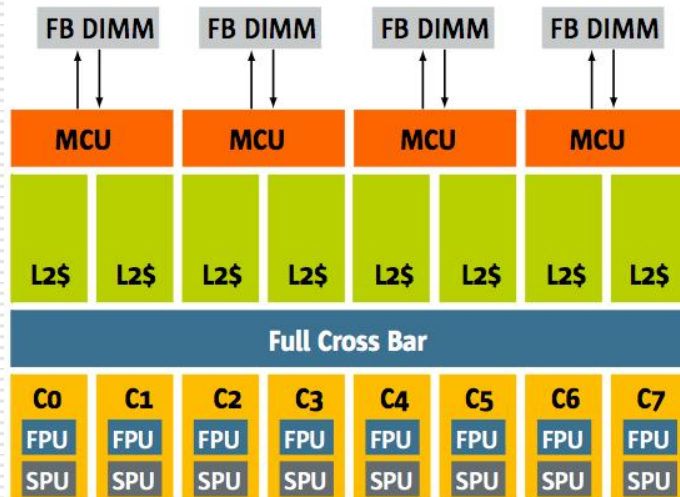
Problems with Multi-threading

- ❑ Cost (time, resources) of switching trades off with work: larger switching cost means more useful work completed before switch ...
instruction level too low?
 - ❑ Need many threads w/o dependences & ...
 - Threads must meet preceding criterion
 - Computations grow & shrink thread count (loop control) implies potential thread starvation
 - Fine-grain threads most numerous, but have least locality
-

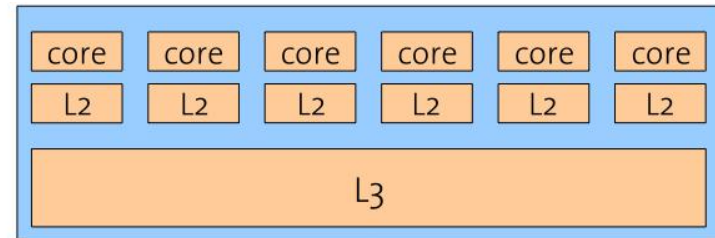
Multi-core Chips

- ❑ Multi-core means more than one processor per chip – generalization of SMT
 - ❑ Consequence of Moore's Law
 - ❑ IBM's PowerPC 2002, AMD Dual Core Opteron 2005, Intel CoreDuo 2006
 - ❑ A small amount of multi-threading included
 - ❑ Main advantage: More ops per tick
 - ❑ Main disadvantages: Programming, BW
-

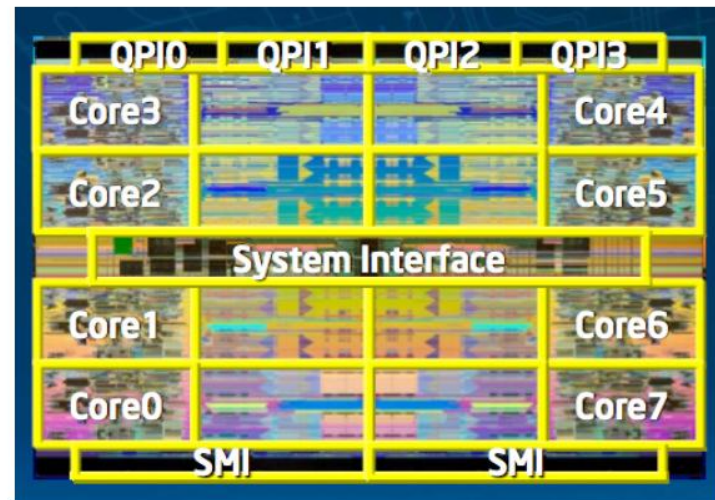
Diversity Among Small Systems



Sun Niagara T2



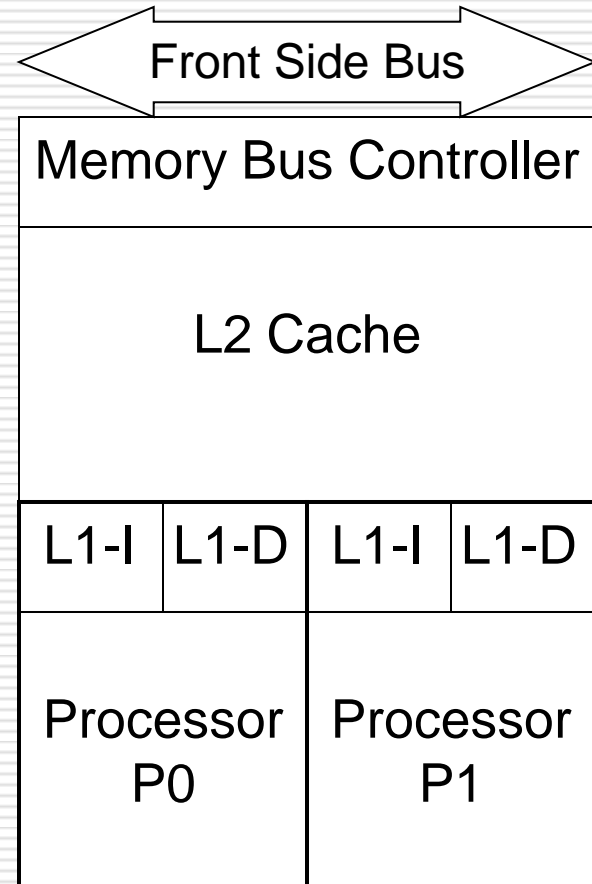
AMD Opteron (Istanbul)



Intel Nehalem (Beckton)

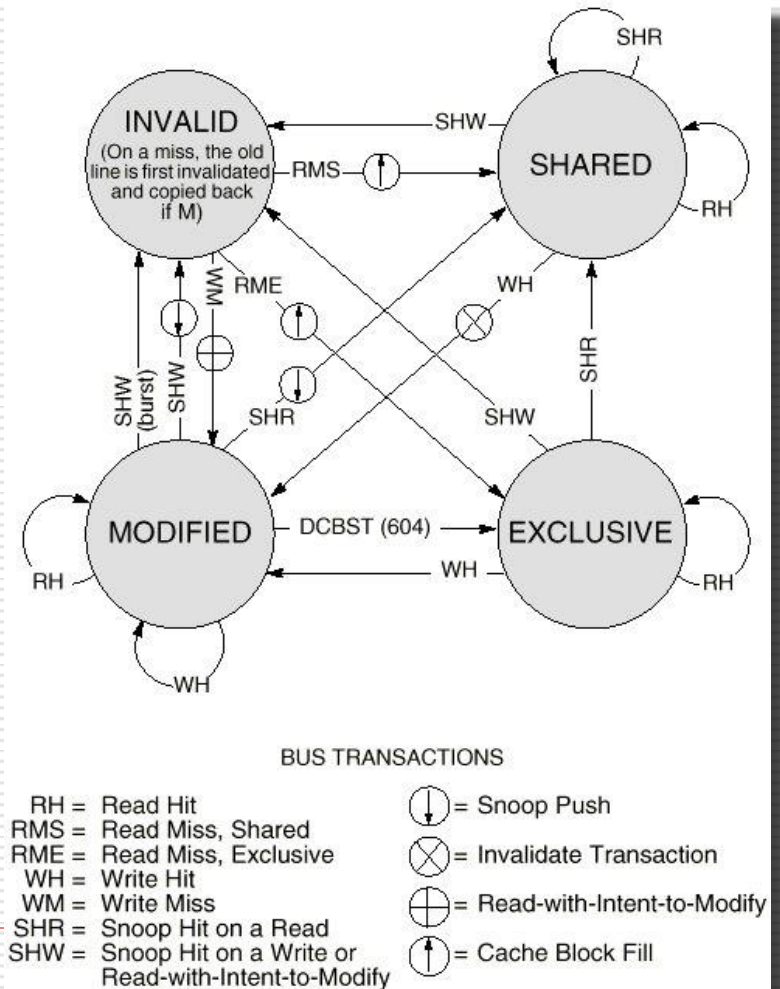
Intel CoreDuo

- ❑ 2 32-bit Pentiums
- ❑ Private 32K L1s
- ❑ Shared 2M-4M L2
- ❑ MESI cc-protocol
- ❑ Shared bus control and memory bus



MESI Protocol

- Standard Protocol for cache - coherent shared memory
- Mechanism for multiple caches to give single memory image
- We will not study it
- 4 states can be amazingly rich



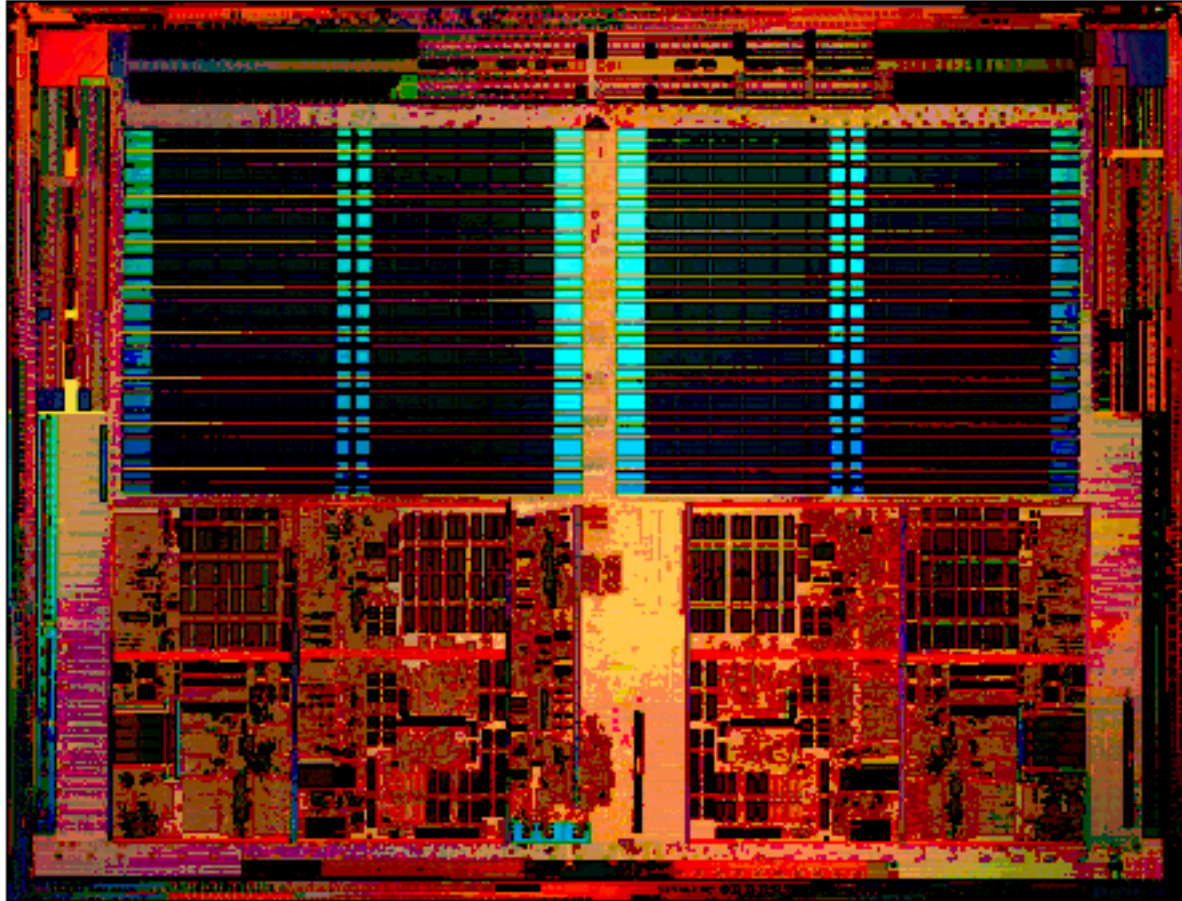
Thanks: Slater & Tibrewala of CMU

Modified
Exclusive
Shared
Invalid

MESI, Intuitively

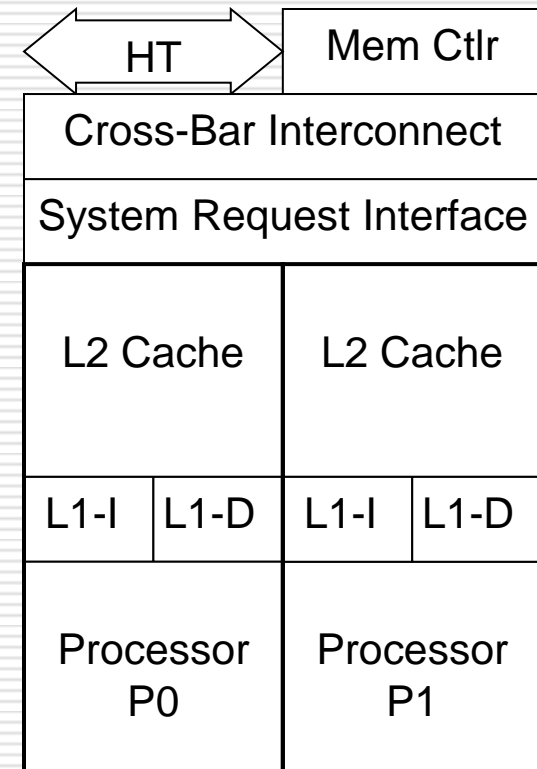
- ❑ Upon loading, a line is marked E, subsequent reads are OK; write marks M
 - ❑ Seeing another load, mark as S
 - ❑ A write to an S, sends I to all, marks as M
 - ❑ Another's read to an M line, writes it back, marks it S
 - ❑ Read/write to an I misses
 - ❑ Related scheme: MOESI (used by AMD)
-

AMD Dual Core Opteron

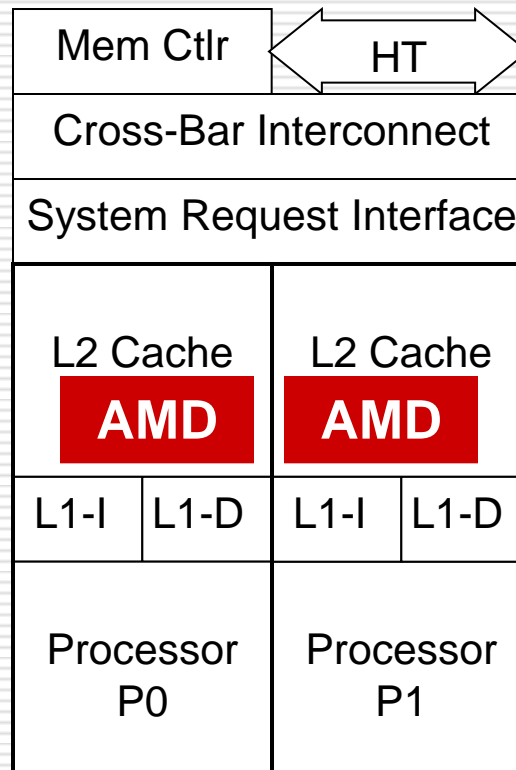
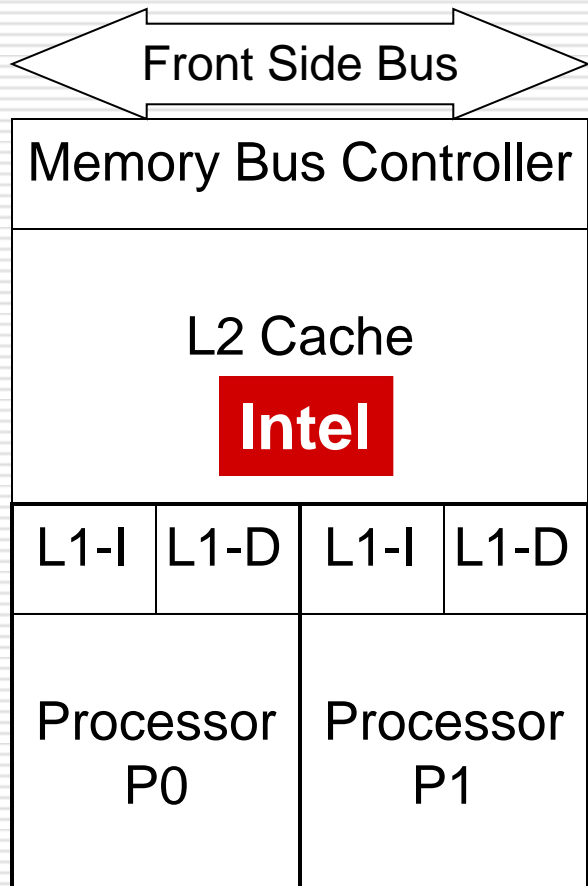


AMD Dual Core Opteron

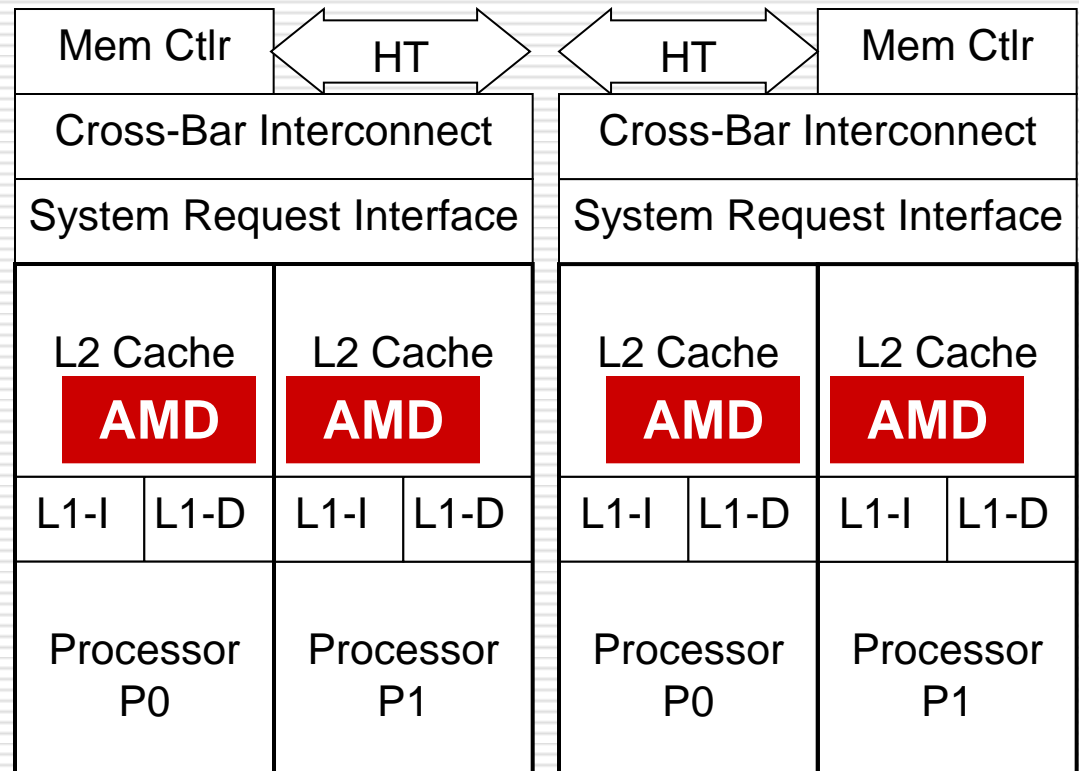
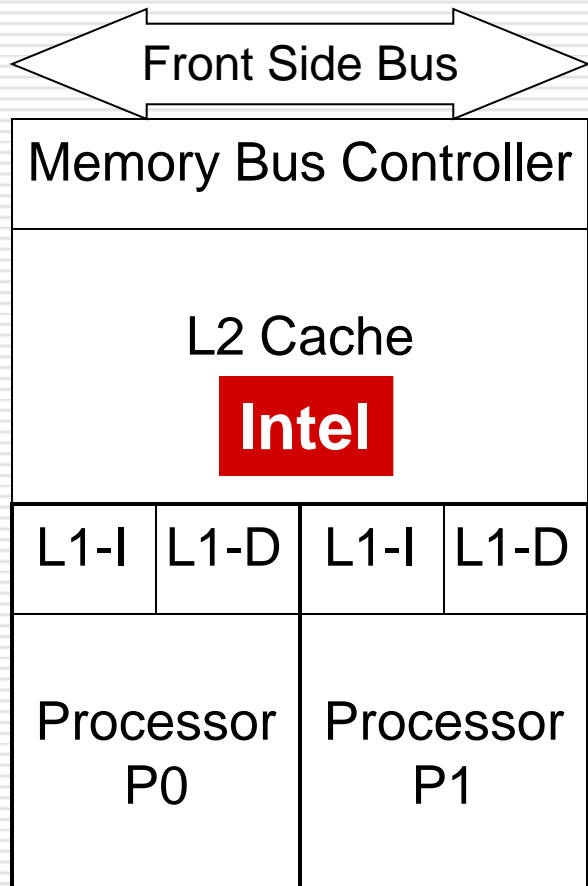
- ❑ 2 64-bit Opterons
- ❑ 64K private L1s
- ❑ 1 MB private L2s
- ❑ MOESI cc-protocol
- ❑ Direct connect shared memory



Comparing Core Duo/Dual Core

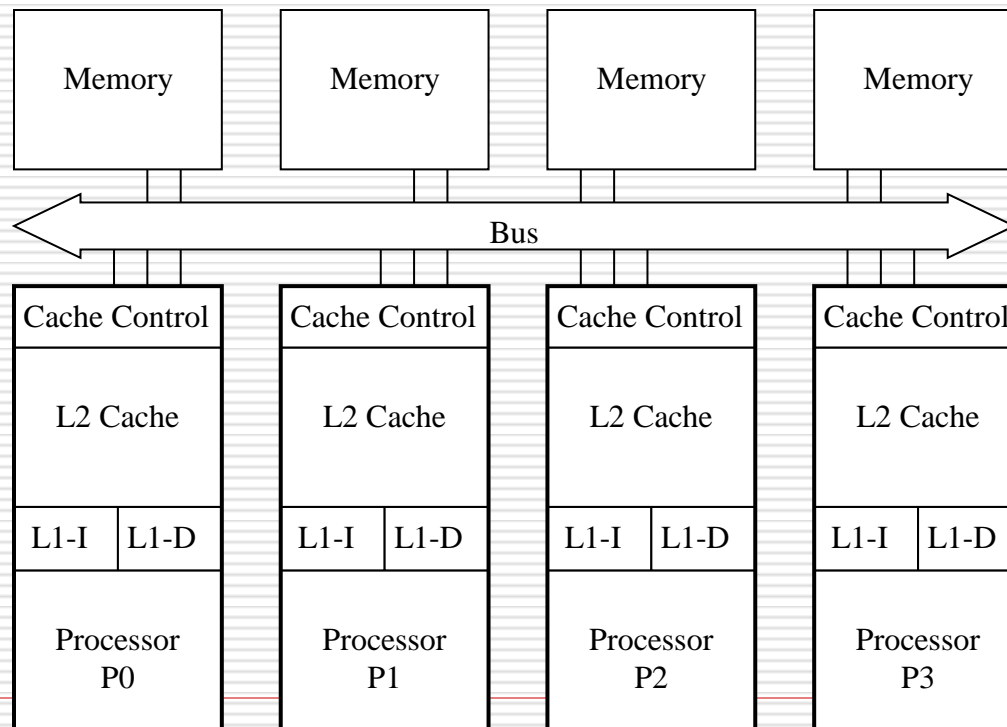


Comparing Core Duo/Dual Core

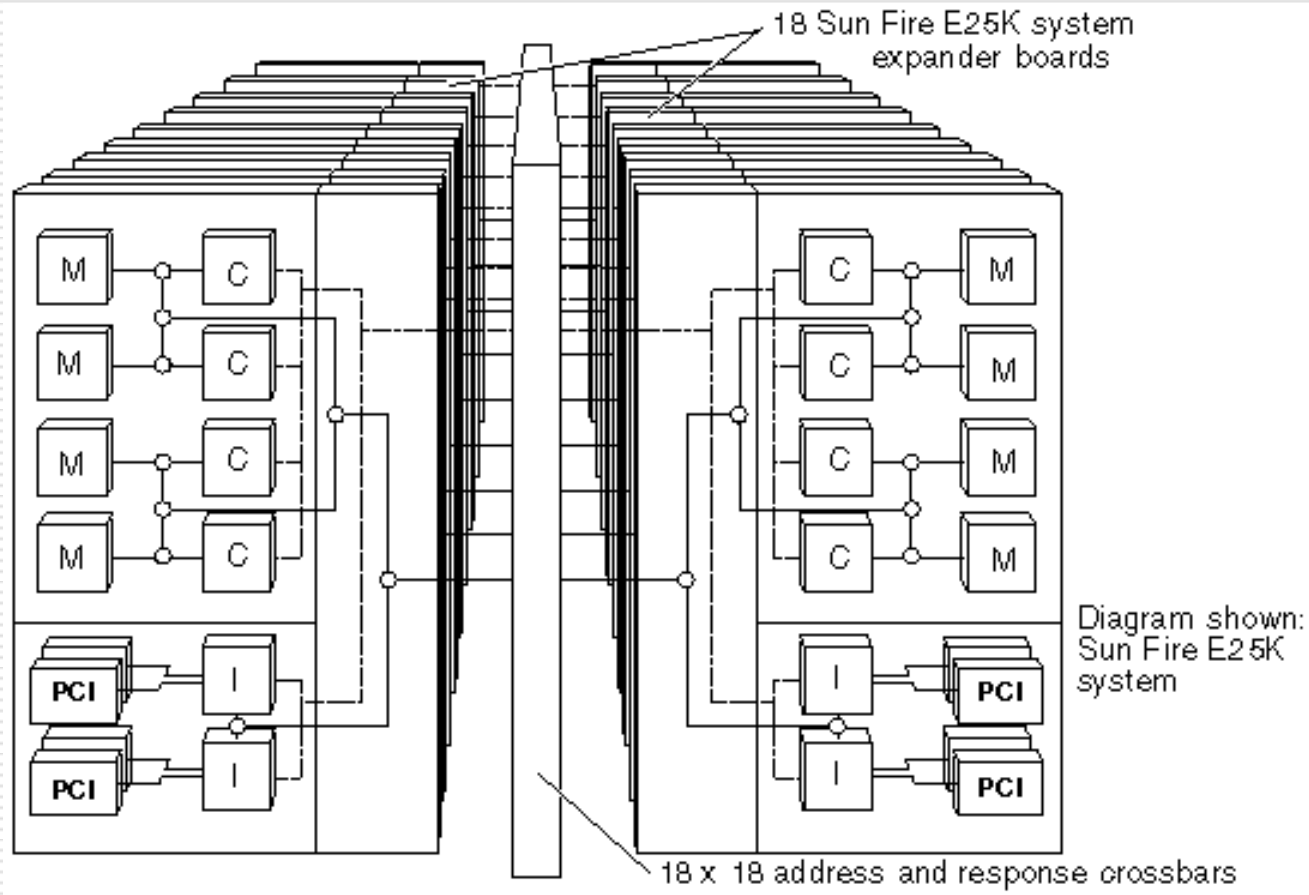


Symmetric Multiprocessor on a Bus

- ❑ The bus is a point that serializes references
- ❑ A serializing point is a shared mem enabler

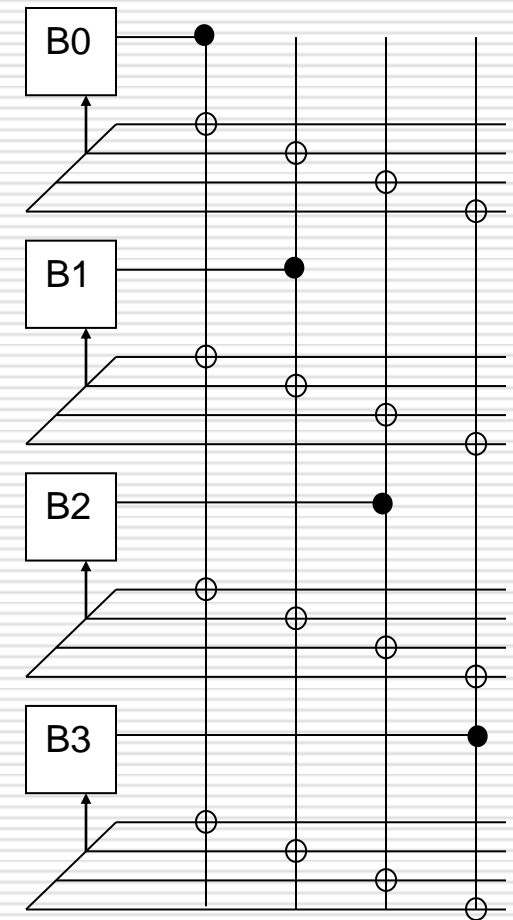


Sun Fire E25K



Cross-Bar Switch

- ❑ A crossbar is a network connecting each processor to every other processor
- ❑ Used in CMU's 1971 C.MMP, 16 proc PDP-11s
- ❑ Crossbars grow as n^2 making them impractical for large n



Sun Fire E25K

- ❑ X-bar gives low latency for snoops allowing for shared memory
 - ❑ 18 x 18 X-bar is basically the limit
 - ❑ Raising the number of processors per node will, on average, increase congestion
 - ❑ How could we make a larger machine?
-

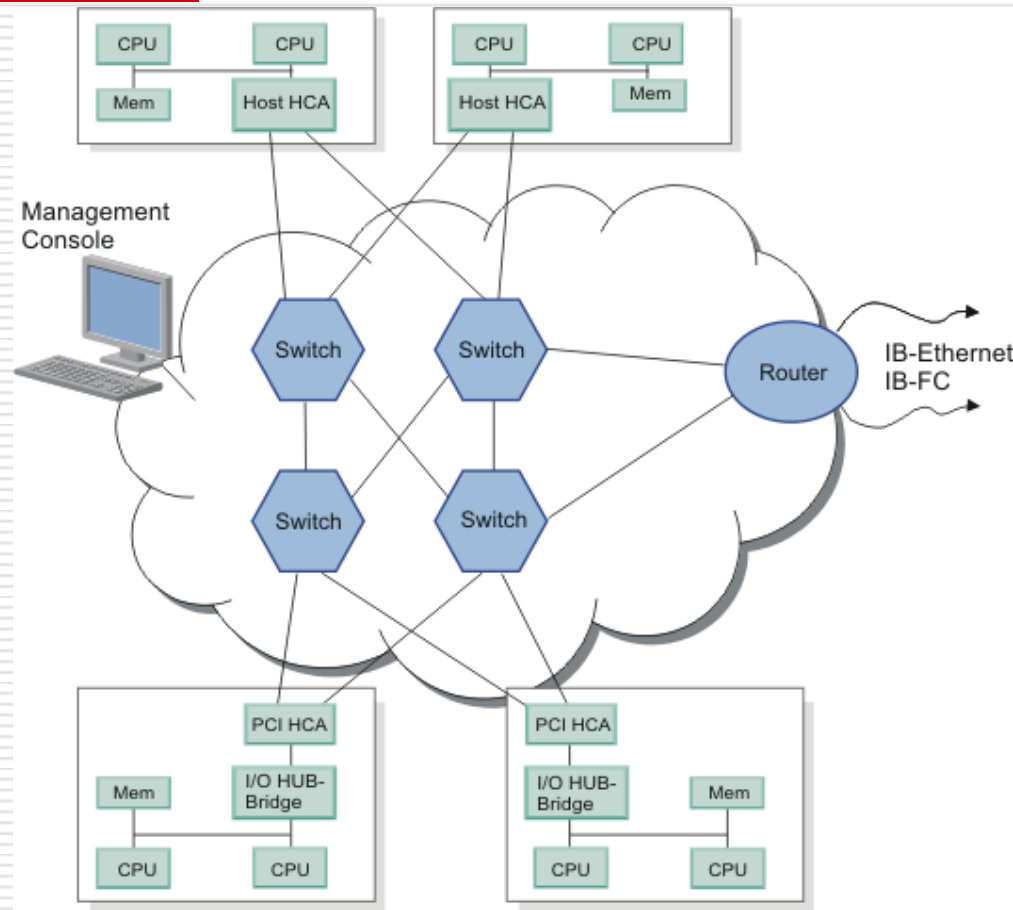
Co-Processor Architectures

- A powerful parallel design is to add 1 or more subordinate processors to std design
 - Floating point instructions once implemented this way
 - Graphics Processing Units - deep pipelining
 - Cell Processor - multiple SIMD units
 - Attached FPGA chip(s) - compile to a circuit

 - These architectures will be discussed later
-

Clusters

- ❑ Interconnecting with InfiniBand
- ❑ Switch-based technology
 - Host channel adapters (HCA)
 - Peripheral computer interconnect (PCI)



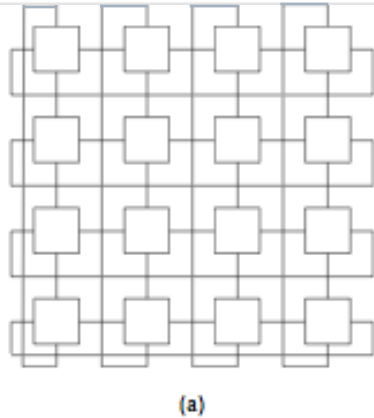
Thanks: **IBM's Clustering systems using InfiniBand Hardware**

Clusters

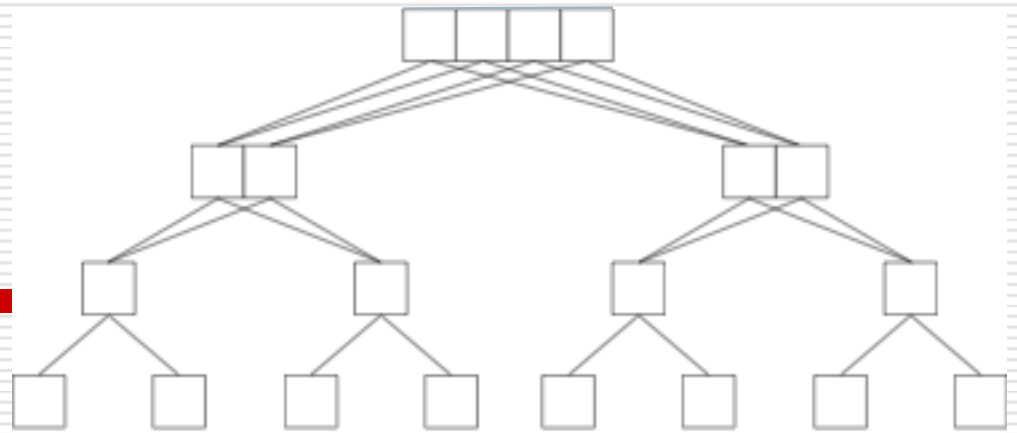
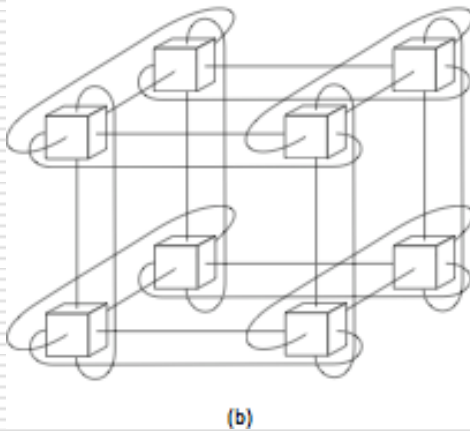
- ☐ Cheap to build using commodity technologies
 - ☐ Effective when interconnect is “switched”
 - ☐ Easy to extend, usually in increments of 1
 - ☐ Processors often have disks “nearby”
 - ☐ No shared memory
 - ☐ Latencies are usually large
 - ☐ Programming uses message passing
-

Networks

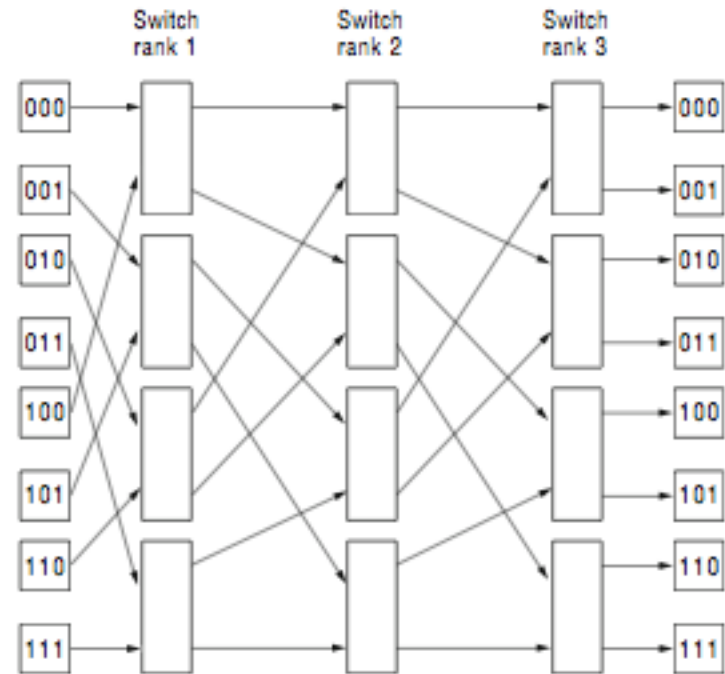
**Torus
(Mesh)**



**Hyper-
Cube**



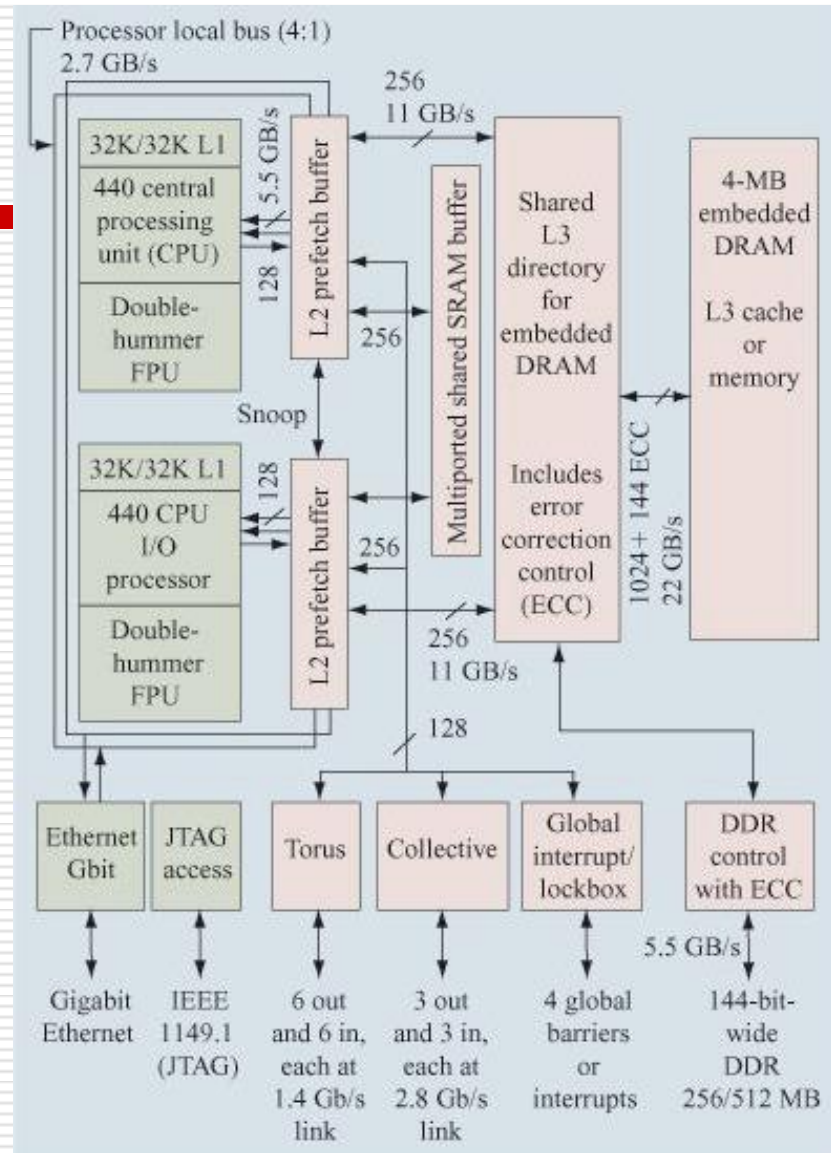
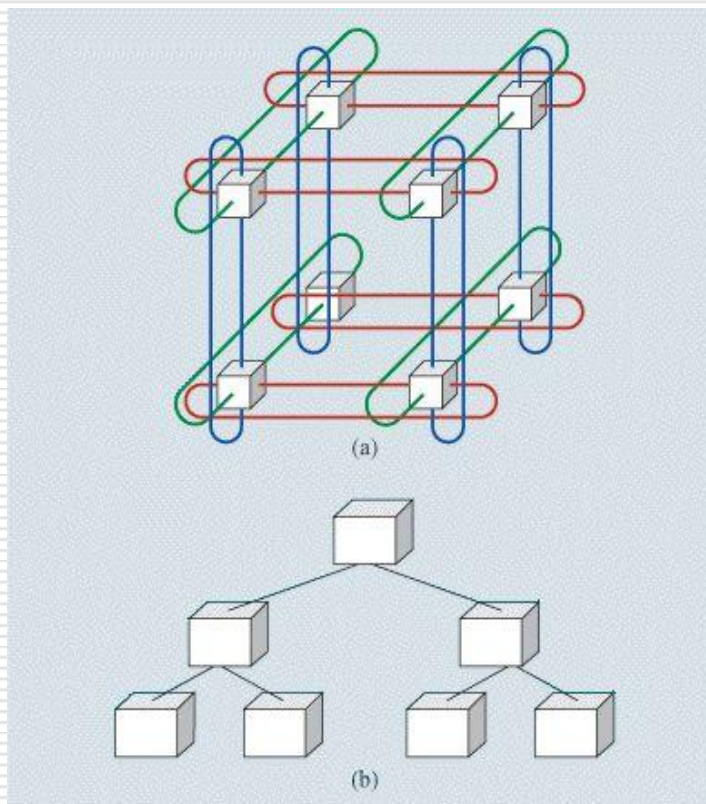
Fat Tree



Omega Network

Supercomputer

□ BlueGene/L



BlueGene/L Specs

- ❑ A 64x32x32 torus = 65K 2-core processors
 - ❑ Cut-through routing gives a worst-case latency of 6.4 μ s
 - ❑ Processor nodes are dual PPC-440 with “double hummer” FPUs
 - ❑ Collective network performs global reduce for the “usual” functions
-

Summarizing Architectures

- ❑ Two main classes
 - Complete connection: CMPs, SMPs, X-bar
 - ❑ Preserve single memory image
 - ❑ Complete connection limits scaling to ...
 - ❑ Available to everyone
 - Sparse connection: Clusters, Supercomputers, Networked computers used for parallelism (Grid)
 - ❑ Separate memory images
 - ❑ Can grow “arbitrarily” large
 - ❑ Available to everyone with air conditioning
 - ❑ Differences are significant; world views diverge
-

Break

- During the break, consider which aspects of the architectures we've seen should be high-lighted and which should be abstracted away
-

The Parallel Programming Problem

- ❑ Some computations can be platform specific
 - ❑ Most should be platform independent
 - ❑ Parallel Software Development Problem:
How do we neutralize the machine differences given that
 - Some knowledge of execution behavior is needed to write programs that perform
 - Programs must port across platforms effortlessly, meaning, by at most recompilation
-

Options for Solving the PPP

- ☐ Leave the problem to the compiler ...

Options for Solving the PPP

- ❑ Leave the problem to the compiler ...
 - Very low level parallelism (ILP) is already being exploited
 - Sequential languages cause us to introduce unintentional sequentiality
 - Parallel solutions often require a paradigm shift
 - Compiler writers' track record over past 3 decades not promising ... recall HPF
 - Bottom Line: Compilers will get more helpful, but they probably won't solve the PPP
-

Options for Solving the PPP

- ☐ Adopt a very abstract language that can target to any platform ...

Options for Solving the PPP

- ☐ Adopt a very abstract language that can target to any platform ...
 - No one wants to learn a new language, no matter how cool
 - How does a programmer know how efficient or effective his/her code is? Interpreted code?
 - What are the “right” abstractions and statement forms for such a language?
 - ☐ Emphasize programmer convenience?
 - ☐ Emphasize compiler translation effectiveness?
-

Options for Solving the PPP

- Agree on a set of parallel primitives (spawn process, lock location, etc.) and create libraries that work w/ sequential code ...

Options for Solving the PPP

- Agree on a set of parallel primitives (spawn process, lock location, etc.) and create libraries that work w/ sequential code ...
 - Libraries are a mature technology
 - To work with multiple languages, limit base language assumptions ... L.C.D. facilities
 - Libraries use a stylized interface (fcn call) limiting possible parallelism-specific abstractions
 - Achieving consistent semantics is difficult
-


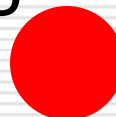


Options for Solving the PPP

- Create an abstract machine model that accurately describes common capabilities and let the language facilities catch up ...
-

Options for Solving the PPP

- Create an abstract machine model that accurately describes common capabilities and let the language facilities catch up ...
 - Not a full solution until languages are available
 - The solution works in sequential world (RAM)
 - Requires discovering (and predicting) what the common capabilities are
 - Solution needs to be (continually) validated against actual experience
-

Summary of Options for PPP

- ☐ Leave the problem to the compiler ... 
 - ☐ Adopt a very abstract language that can target to any platform ... 
 - ☐ Agree on a set of parallel primitives (spawn process, lock location, etc.) and create libraries that work w/ sequential code ... 
 - ☐ Create an abstract machine model that accurately describes common capabilities and let the language facilities catch up ... 
-

Why is Seq Programming Successful

When we write programs in C they are ...

- **Efficient** -- programs run fast, especially if we use performance as a goal
 - traverse arrays in row major order to improve caching
- **Economical** -- use resources well
 - represent data by packing memory
- **Portable** -- run well on any computer with C compiler
 - all computers are universal, but with C fast programs are fast everywhere
- **Easy to write** -- we know many 'good' techniques
 - reference data, don't copy

These qualities all derive from von Neumman model

Von Neumann (RAM) Model

- ❑ Call the ‘standard’ model of a random access machine (RAM) the von Neumann model
 - ❑ A processor interpreting 3-address instructions
 - ❑ PC pointing to the next instruction of program in memory
 - ❑ “Flat,” randomly accessed memory requires 1 time unit
 - ❑ Memory is composed of fixed-size addressable units
 - ❑ One instruction executes at a time, and is completed before the next instruction executes
- ❑ The model is not literally true, e.g., memory is hierarchical but made to “look flat”

C directly implements this model in a HLL

Why Use Model That's Not Literally True?

- ❑ Simple is better, and many things--GPRs, floating point format--don't matter at all
- ❑ Avoid embedding assumptions where things could change ...
 - Flat memory, tho originally true, is no longer right, but we don't retrofit the model; we don't want people "programming to the cache"
 - ❑ Yes, exploit spatial locality
 - ❑ No, avoid blocking to fit in cache line, or tricking cache into prefetch, etc.
 - Compilers bind late, particularize and are better than you are!

vN Model Contributes To Success

- ❑ The cost of C statements on the vN machine is “understood” by C programmers ...
- ❑ How much time does $A[r][s] += B[r][s];$ take?
 - ❑ Load row_size_A, row_size_B, r, s, A_base, B_base (6)
 - ❑ tempa = (row_size_A * r + s) * data_size (3)
 - ❑ tempb = (row_size_B * r + s) * data_size (3)
 - ❑ A_base + tempa; B_base + tempb; load both values (4)
 - ❑ Add values and return to memory (2)
- Same for many operations, any data size
- ❑ Result is measured in “instructions” not time

Widely known and effectively used

Portability

- Most important property of the C-vN coupling:
It is approximately right everywhere
- Why so little variation in sequential computers?



**HW vendors must run
installed SW so follow
vN rules**

**SW vendors must run
on installed HW so
follow vN rules**

**Everyone wins ... no
motive to change**

Von Neumann Summary

- ❑ The von Neumann model “explains” the costs of C because C expresses the facilities of the von Neumann machines in programming terms
- ❑ Knowing the relationship between C and the von Neumann machine is essential for writing fast programs
- ❑ Following the rules produces good results everywhere because everyone benefits
- ❑ These ideas are “in our bones” ... it’s how we think

What is the parallel version of vN?

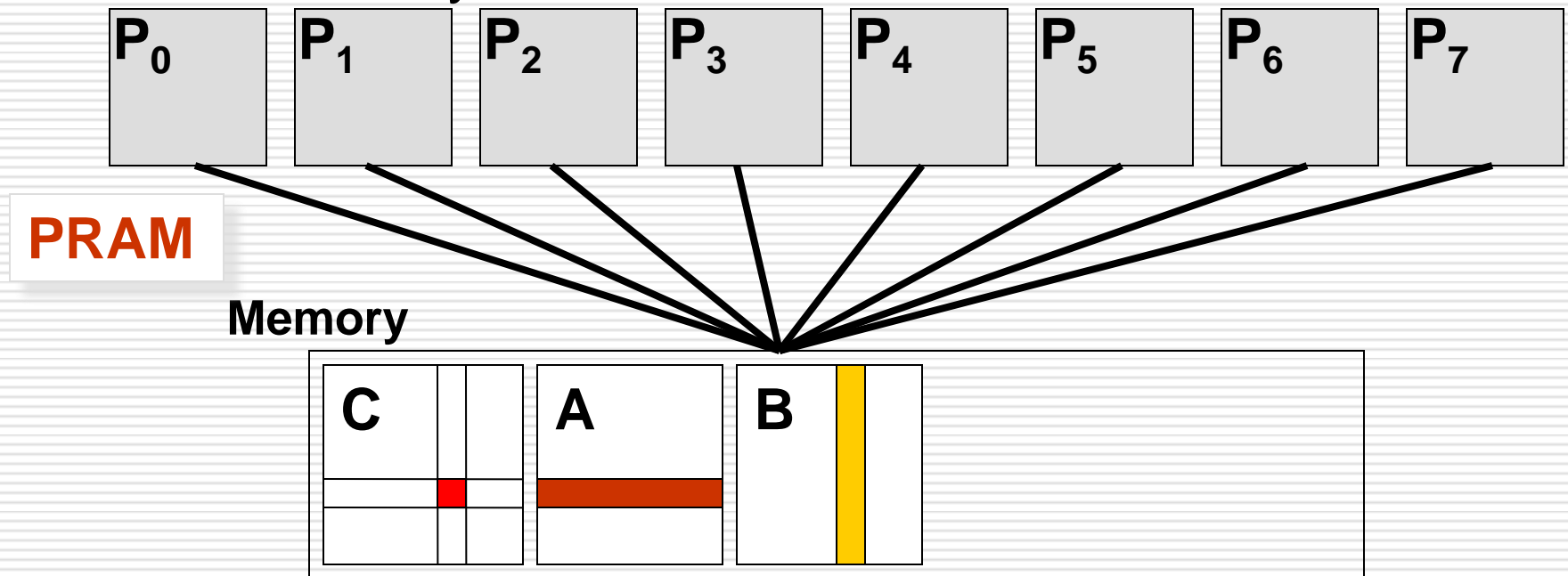
PRAM Often Proposed As A Candidate

- ❑ PRAM (Parallel RAM) ignores memory organization, collisions, latency, conflicts, etc.
- ❑ Ignoring these are *claimed* to have benefits ...
 - Portable everywhere since it is very general
 - It is a simple programming model ignoring only insignificant details -- off by “only log P”
 - Ignoring memory difficulties is OK because hardware can “fake” a shared memory
 - Good for getting started: Begin with PRAM then refine the program to a practical solution if needed

Recall Parallel Random-Access Machine

PRAM has any number of processors

- Every proc references any memory in “time 1”
- Memory read/write collisions must be resolved



SMPs implement PRAMs for small P ... not scalable⁵⁶

Variations on PRAM

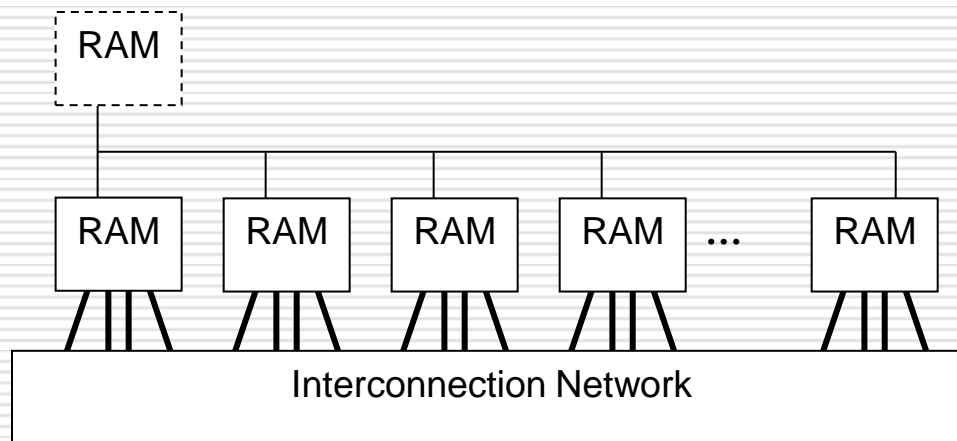
Resolving the memory conflicts considers read and write conflicts separately

- ❑ Exclusive read/exclusive write (EREW)
 - ❑ The most limited model
- ❑ Concurrent read/exclusive write (CREW)
 - ❑ Multiple readers are OK
- ❑ Concurrent read/concurrent write (CRCW)
 - ❑ Various write-conflict resolutions used
- ❑ There are at least a dozen other variations

All theoretical -- not used in practice

CTA Model

- Candidate Type Architecture: A model with P standard processors, d degree, λ latency



- Node == processor + memory + NIC

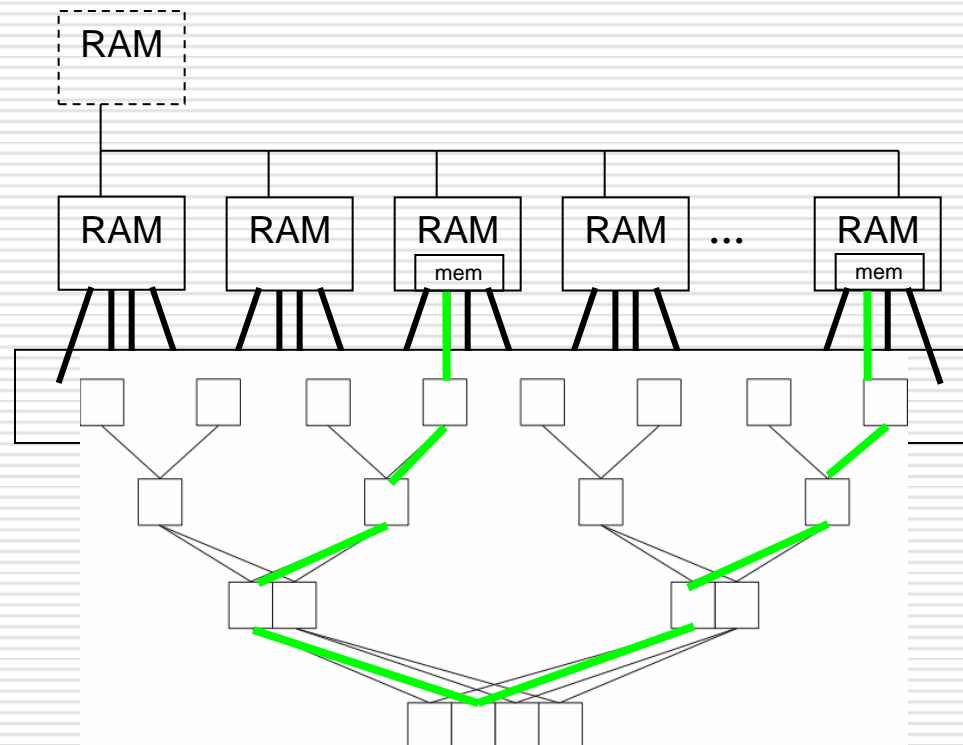
Key Property: Local memory ref is 1, global memory is λ

What CTA Doesn't Describe

- ❑ CTA has no global memory ... but memory could be globally *addressed*
 - ❑ Mechanism for referencing memory not specified: shared, message passing, 1-side
 - ❑ Interconnection network not specified
 - ❑ λ is not specified beyond $\lambda \gg 1$ -- cannot be because every machine is different
 - ❑ Controller, combining network “optional”
-

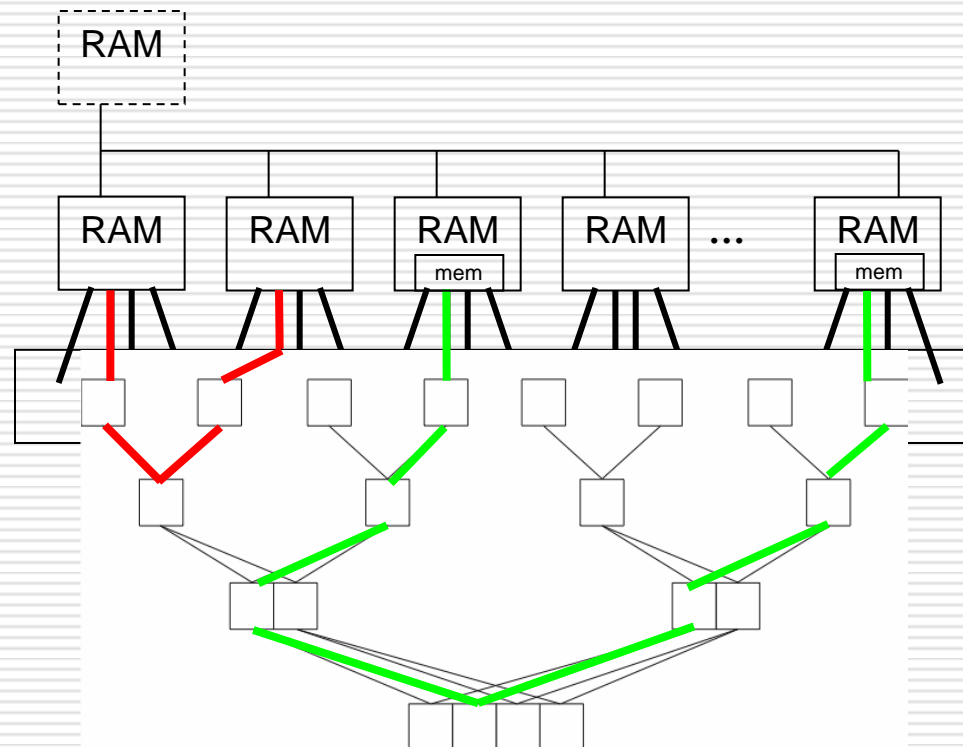
More On the CTA

- Consider what the diagram means...



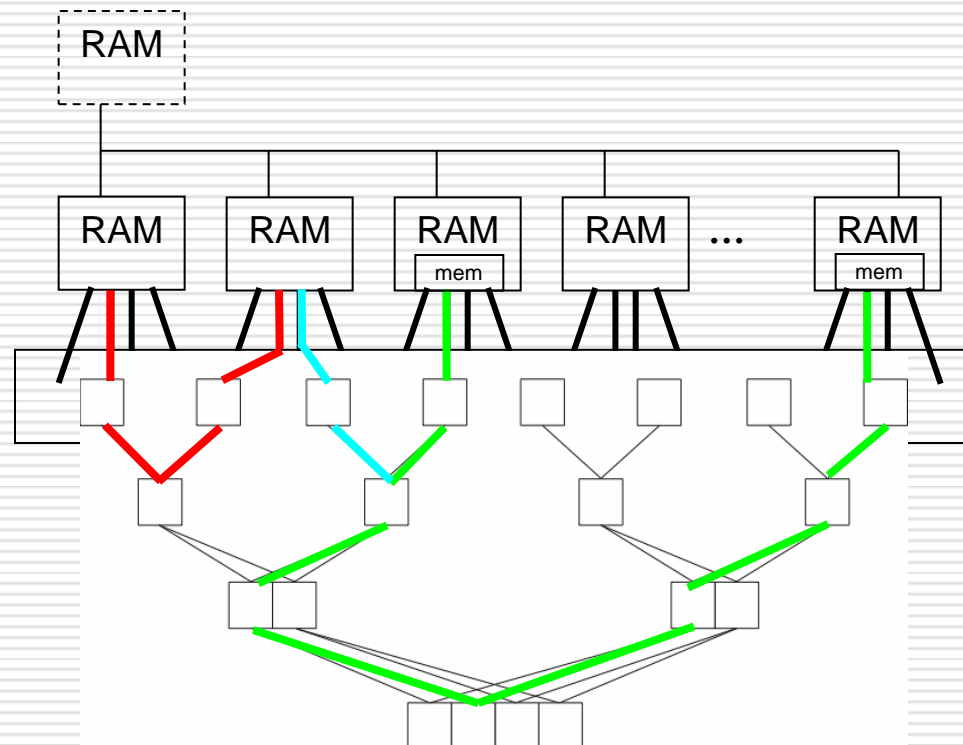
More On the CTA

- Consider what the diagram means...



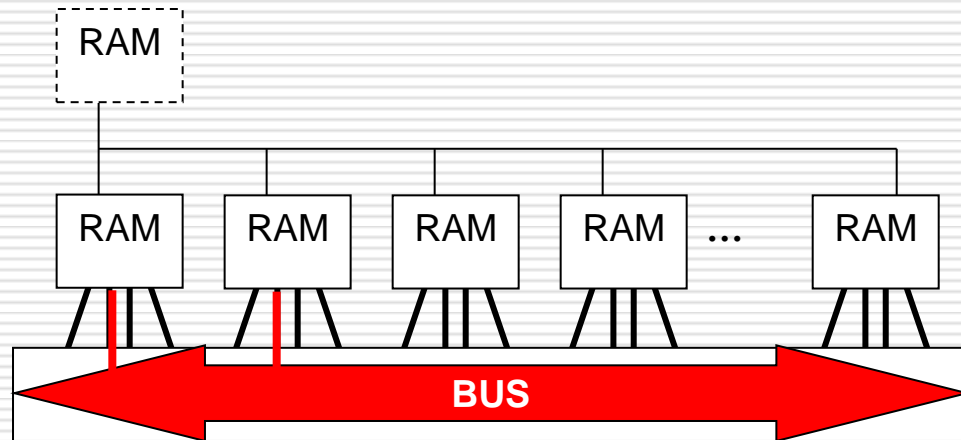
More On the CTA

- Consider what the diagram means...



More On the CTA

- Consider what the diagram **doesn't** mean...



- After ACKing that CTA doesn't model buses, accept that it's a good first approx.
-

Typical Values for λ

- ❑ Lambda can be estimated for any machine (given numbers include **no** contention or congestion)

CMP	AMD	100
SMP	Sun Fire E25K	400-660
Cluster	Itanium + Myrinet	4100-5100
Super	BlueGene/L	5000

} Lg λ range
=> cannot
be ignored

As with merchandizing: **It's location, location, location!**

Measured Numbers

□ Values (approximating) λ for small systems

System	Cache	Latency cycles	Throughput msgs/kcycle
2×4-core Intel	shared	180	11.97
	non-shared	570	3.78
2×2-core AMD	same die	450	3.42
	one-hop	532	3.19
4×4-core AMD	shared	448	3.57
	one-hop	545	3.53
	two-hop	659	3.19
8×4-core AMD	shared	538	2.77
	one-hop	613	2.79
	two-hop	682	2.71

Communication Mechanisms

□ Shared addressing

- One consistent memory image; primitives are load and store
- Must protect locations from races
- Widely considered most convenient, though it is often tough to get a program to perform
- CTA implies that best practice is to keep as much of the problem private; use sharing only to communicate

A common pitfall: **Logic is too fine grain**

Communication Mechanisms

☐ Message Passing

- No global memory image; primitives are `send()` and `recv()`
- Required for most large machines
- User writes in sequential language with message passing library:
 - ☐ Message Passing Interface (MPI)
 - ☐ Parallel Virtual Machine (PVM)
- CTA implies that best practice is to build and use own abstractions

Lack of abstractions makes message passing brutal

Communication Mechanisms

□ One Sided Communication

- One global address space; primitives are `get()` and `put()`
- Consistency is the programmer's responsibility
- Elevating mem copy to a comm mechanism
- Programmer writes in sequential language with library calls -- not widely available unfortunately
- CTA implies that best practice is to build and use own abstractions

One-sided is lighter weight than message passing

Programming Implications

- How does CTA influence programming ...
 - Discuss
 - Expression evaluation: Same/Different?
 - Relationship among processors?
 - Data structures?
 - Organization of work?
 - ...
-

Find Maximum in Parallel (Valiant)

Task: Find largest of n integers w/ n processors

Model: CRCW PRAM (writes OK if same value)

How would YOU do it?

L.G.Valiant, “Parallelism in comparison problems,” SIAM J. Computing 4(3):348-355, 1975

L.G. Valiant, “A Bridging Model for Parallel Computation,” CACM 33(8):103-111, 1990

R.J. Anderson & L. Snyder, “A Comparison of Shared and Nonshared Memory Models for Parallel Computation,” *Proc. IEEE* 79(4):480-487

Algorithm Sketch

Algorithm: T rounds of $O(1)$ time each

In round, process groups of m vals, v_1, v_2, \dots, v_m

- Fill m memory locations x_1, x_2, \dots, x_m with 1s to be “knocked out”
- For each $1 \leq i, j \leq m$ a processor tests ...
 $\text{if } v_i < v_j \text{ then } x_i = 0 \text{ else } x_j = 0$
- If $x_k = 1$ it's max of group; pass v_k to next round

The ‘trick’ is to pick m right to minimize T

Finding Max (continued)

Round 1: $m = 3$

Input

v_1	v_2	v_3
20	3	34

Schedule

	v_1	v_2	v_3
v_1	-	$v_1:v_2$	$v_1:v_3$
v_2	-	-	$v_2:v_3$
v_3	-	-	-

For groups of size 3, three tests can find max, i.e. 3 procesors

Knock out

x_1	x_2	x_3
1	1	1



x_1	x_2	x_3
0	0	1

Output

Solving Whole Problem

- ❑ Round 1 uses P processors to find the max in groups of $m=3$... producing $P/3$ group maxes
- ❑ Round 2 uses P processors to find the max in groups of $m=7$... producing $P/21$ group maxes
- ❑ Generally to find the max of a group requires $m(m-1)/2$ comparisons
- ❑ Picking m when there are P processors, r maxes ... largest m s.t. $(r/m)(m(m-1)/2) \leq P$ i.e. $r(m-1) \leq 2P$

Finding Max (continued)

- Initially, $r = P$, so
 $r(m-1) \leq 2P$
implies $m = 3$, producing $r = P/3$
- For $(P/3)(m-1) \leq 2P$ implies next group = 7
- Etc.
- Group size increases quadratically implying the maximum is found in $O(\log \log n)$ steps on CRCW PRAM

It's very clever, but is it of any practical use?

Assessing Valiant's Max Algorithm

The PRAM model caused us to ...

- Exploit the “free use” of read and write collisions, which are not possible in practice
- Ignore the costs of data motion, so we adopt an algorithm that runs *faster* than the time required to bring all data values together, which is $\Omega(\log n)$
- So what?

Running Valiant's Algorithm

- ❑ PRAM's don't exist and can't be built
- ❑ To run the algorithm we need a simulator for the CRCWPRAM
- ❑ In order to simulate the concurrent reads and the concurrent writes, a parallel computer will need $\Omega(\log P)$ time per step, though there are bandwidth requirements and serious engineering problems to attain that goal
- ❑ *Observed* performance of Valiant's Max:
 $O(\log n \log \log n)$





Alternative Solution

- ❑ What is the best way of computing max using the CTA?
 - A tree algorithm, a variation on global sum
 - $O(\log P)$ time on P processors
 - The tree algorithm doesn't need to be simulated ... it runs in the stated time directly on all existing parallel processors
- ❑ Since $O(\log n) < O(\log n \log \log n)$ the PRAM model mispredicted the best practical algorithm

The PRAM didn't help, it hurt our effort

Is The PRAM A Good Abstraction?

Different Opinions ...

- ☐ OK for finding theoretical limits to parallelism 
- ☐ It is a simple programming model ignoring only insignificant details -- off only by $\log P$ 
- ☐ Ignoring memory difficulties is OK because hardware can “fake” a shared memory 
- ☐ Start with PRAM then evolve to more realistic solution -- good for getting started 

Apply CTA to Count 3s

- How does CTA guide us for Count 3s pgm
 - Array segments will be allocated to local mem
 - Each processor should count 3s in its segment
 - Global total should be formed using reduction
 - Performance is
 - Full parallelism for local processing
 - $\lambda \log n$ for combining
 - Base of log should be large, i.e high degree nodes
 - Same solution as before, but by different rt
-

Summary

- Parallel hardware is a critical component of improving performance through ||-ism ... but there's a Catch-22
 - To have portable programs, we must abstract away from the hardware
 - To write performant programs requires that we respect the hardware realities
- Solve the problem with CTA -- an abstract machine with just enough (realizable) detail to support critical programming decisions

Assignment for Next Time

- Thinking of XML trees, which are made up of well-nested, user-defined matching tags, use the CTA to sketch the logic of a || algorithm to check if an XML file (is / is not) well nested and estimate its performance
 - Simplifications
 - Linear sequence of: (, x,) as in ((xxx)x(x)(xx))
 - Explain the algorithm to a *person*, e.g. a TA grader, giving data allocation, communication specifics, protocol for processor interactions, etc.
 - Assume $n \gg P$, comm costs l , give performance
-