Architecture and Parallel Computers

Chris Kauffman

CS 499: Spring 2016 GMU

Logistics

Reading: Grama Ch 2

- Focus on 2.3-5, material pertaining to distributed memory
- We will return to shared memory arch later in the course
- Cache Coherence, PRAM models, False Sharing, Memory Bus are all shared memory topics
- Sections 2.1 and 2.2 optional, deeper architectures
- Sections 2.6 and 2.7 encouraged, deeper on networks

Assignment 1

- Will post over the weekend
- 8-day turn around
- Mostly written assignment
- Feelings on group work?

The Dining "Swansons" (Philosopher)

- Whole model is premised on limited use: eventually a Swanson with 2 forks will give them up and wait a while before trying to reacquire
- Several Solutions Exist to avoid deadlock

Dijkstra

Number forks, everyone tries to get lower number first

Source: Aditya Y. Bhargava,
Originally: Dustin D'Arnault

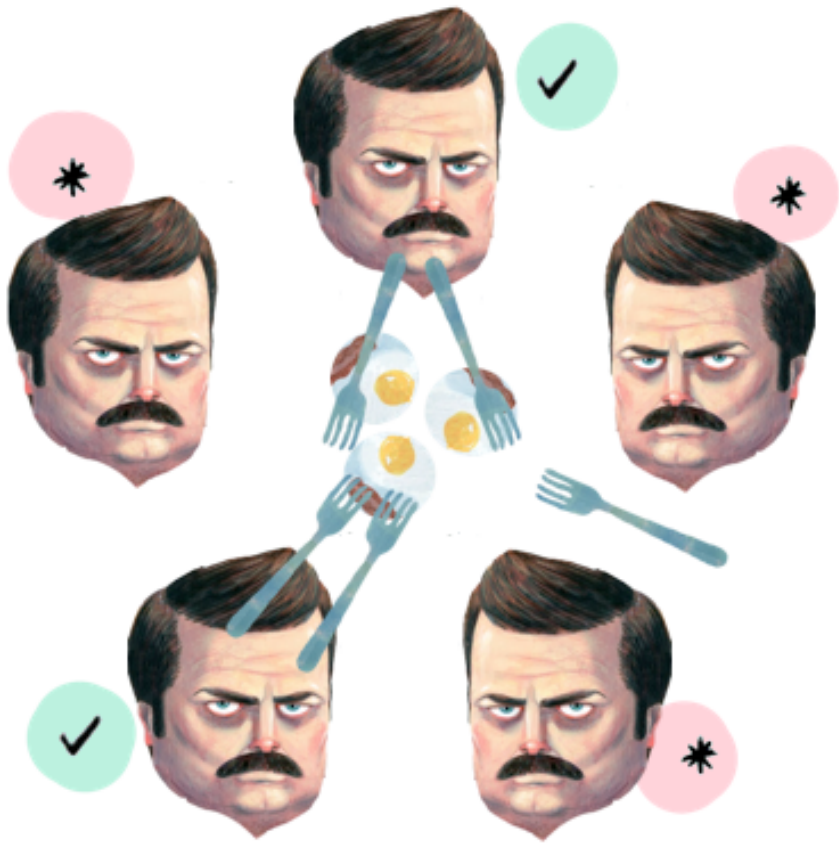
Dining Philosophers: Other Solutions

Waiter Mutex

Obtain "permission" (lock) to pick up forks. Only Swanson can pick up forks at a time. Attempt to pick up both. On failure, relinquish lock. (Locks: After Spring Break)

Chandy/Misra

Requires communication between Swansons. "I want your fork." "No. It's clean and I'm using it." "Fine, I'll wait." "I'm done it's dirty but I wiped it off for you." "Thank you." "I want your fork too." "Mine's already dirty but I'll clean it and it's yours."



Source: Aditya Y. Bhargava,
Originally: Dustin D'Arnault

SISD, SIMD, MIMD, SPAM, and other 4-letter words

- Traditional CPU, Single Instruction Single Data (SISD)

```
ADD r1, r2        # add int in r2 to r1
```

- Most computers now have cpu instructions to add multiple

```
PHADD mm1, mm2  # add two ints in mm2 to ints in mm1
```

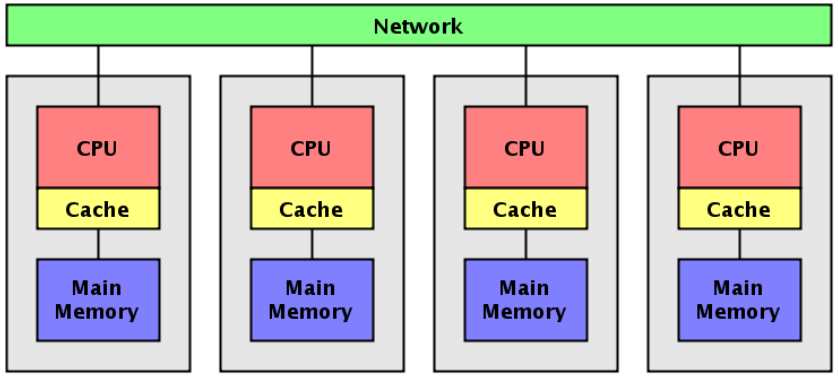
- Low level parallelism good for multimedia stuff/graphics/games
- Flynn's taxonomy discusses several variants

SISD	SIMD	SPMD
MISD	MIMD	MPMD

- Some parallel programs exist as Multiple Program Mulitple Data (MPMD) like client server models
- Our focus and the most common type of parallel program: Single Program Multiple Data (SPMD): *Write one program which processes different hunks of data in parallel*

Recall: Distributed vs Shared Memory

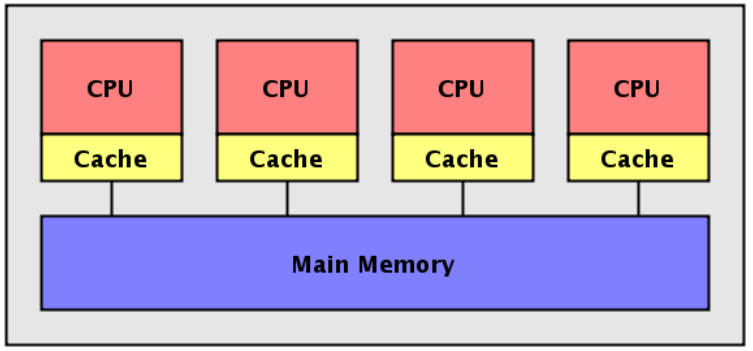
Distributed Memory



Source: Kaminsky/Parallel Java

- Far more scalable/cost effective
- Sharing information requires explicit send/receive commands between processors
- Communication requires more care/more expensive

Shared Memory



Source: Kaminsky/Parallel Java

- Convenience: no explicit send/receive, write shared memory address
- Requires coordination to prevent corrupting memory
- Communication cost is low but requires discipline

Modeling Distributed Memory Parallel Computers

- Will spend a some time discussing networks used in parallel computing
- These have consequences for algorithms, but unless you're building your own machine (for like $1M) you're stuck with what you get

Static Networks for Distributed Machines

- String up a bunch of processing elements (PEs)
- Which PE is connected to which other?
- This can affect the cost of communication

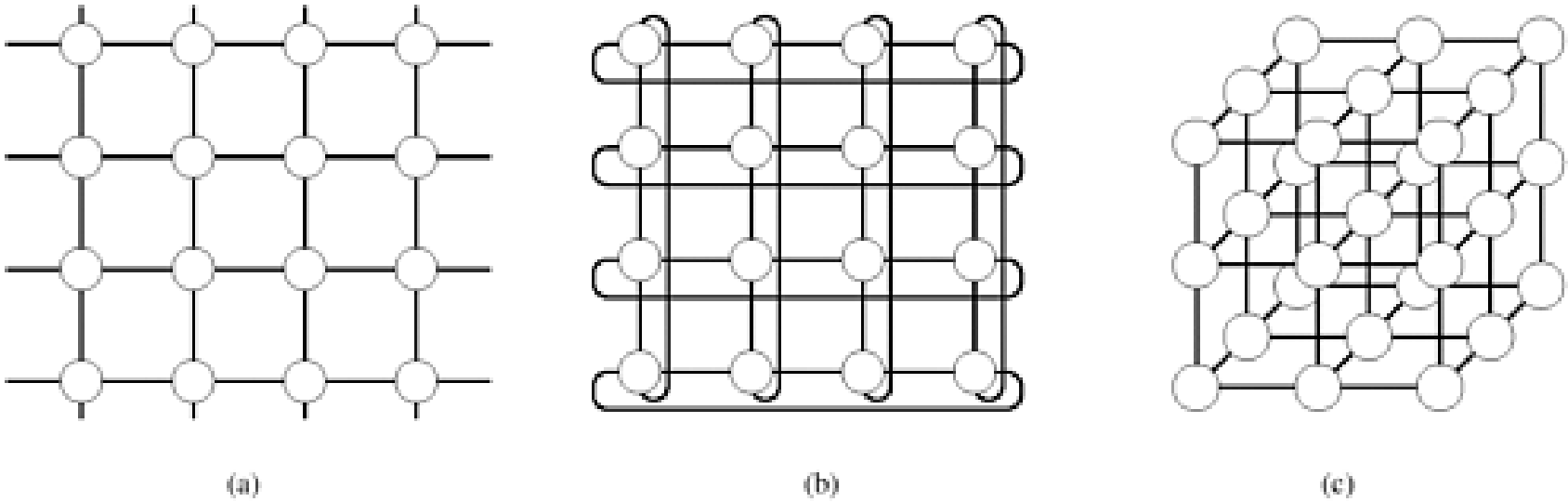
Communication Costs

When sending a message of size m words of memory

- t_s: Startup time, incurred once
- t_h: Per-hop time, overhead incurred for each link between source and destination
- t_w: Per-word transfer time between two nodes, takes $t_w \times M$ time for each link between source and destination
- L: number of links to traverse
- M: number of words being sent
- Typical model for communication time w/ packet routing

$$t_{comm} = t_s + Lt_h + t_w M$$

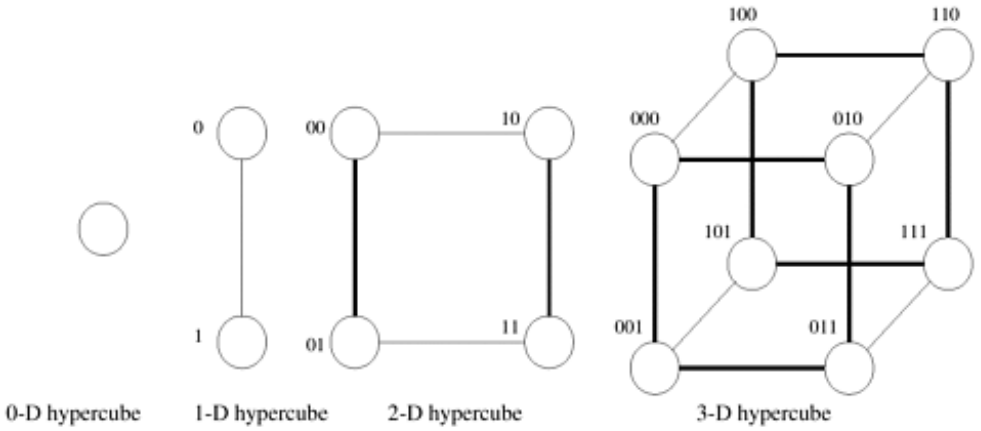
Grid and Torus

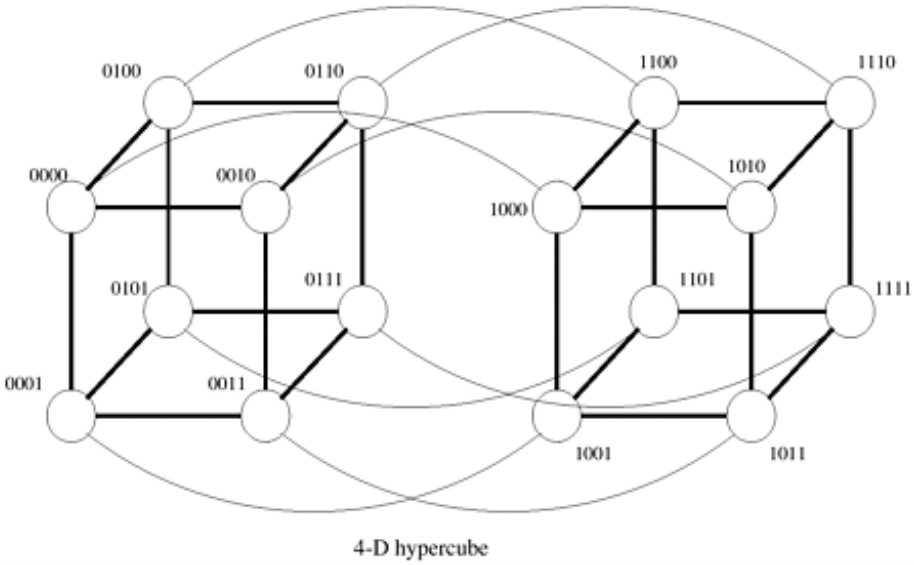


- Common arrangement of links between PEs
- Each PE node connected to neighbors
- When wrapping around, grid becomes a torus
- For a 2D torus with p nodes, how many links are required?
- *Hint: surprisingly simple, think of each processor "owning" down and right links*
- How many links in a 3D torus?

HyperCube

- n-dimension hypercube: connect two $(n-1)$ dimension hypercubes, link corresponding nodes
- How many nodes and links in an n-dimension hypercube?
- *Hint: Nodes are easy, links are tricky, try your textbook...*





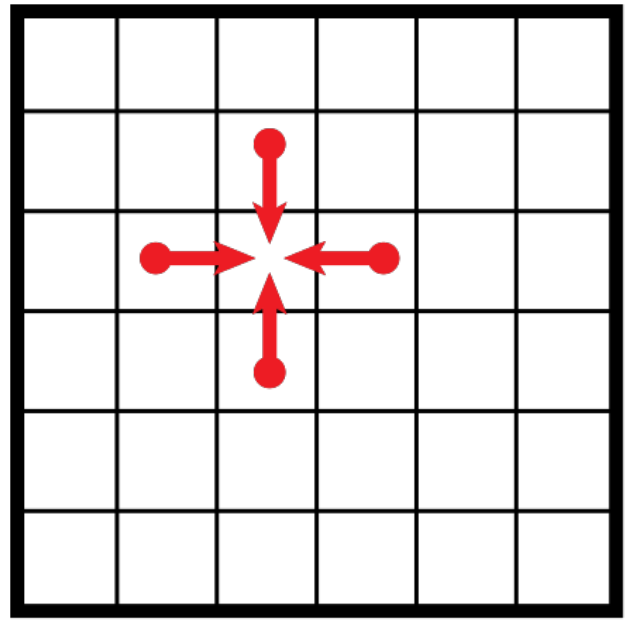
Compare Networks: Parallel Stencil

- p processors
- $\log_2(p)$-dimension Hypercube: $(p \log_2(p)/2)$ links
- 2D-torus: $2p$ links
- Discuss advantages/disadvantages of torus vs hypercube arrangement for this application
- Outline an algorithm, estimate cost-effectiveness

Image "blurring"

- A large image is distributed across the p processors
- Each proc holds a 2D hunk of the image
- To blur the entire image, must assign RGB values which are average of "neighborhood"

Stencil



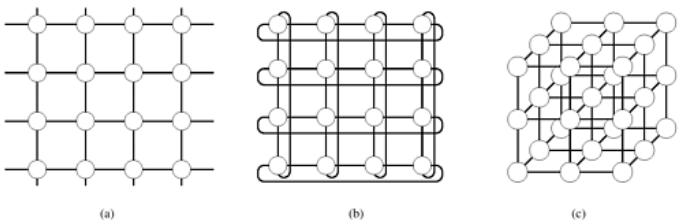
Compare Networks: Parallel Sum

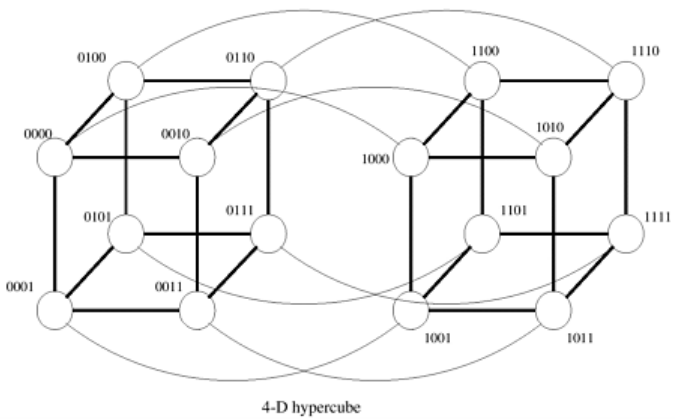
- p processors
- $\log_2(p)$-dimension Hypercube: $(p \log_2(p)/2)$ links
- 2D-torus: $2p$ links
- Discuss advantages/disadvantages of torus vs hypercube arrangement for this application
- Outline an algorithm, estimate cost-effectiveness

Sum Array of Numbers

- Each proc holds a hunk of the data array
- Want a single processor to eventually contain sum o
- State your algorithm: Try to minimize communication at each step, exploit as much parallelism as possible

Networks





4-D hypercube

Some details on Parallel Sum

- We will talk more about parallel sum later
- Parallel sum is an example of a reduction
- For those curious, have a look at Lecture notes by Susan Hayes

Characteristics of Various Networks

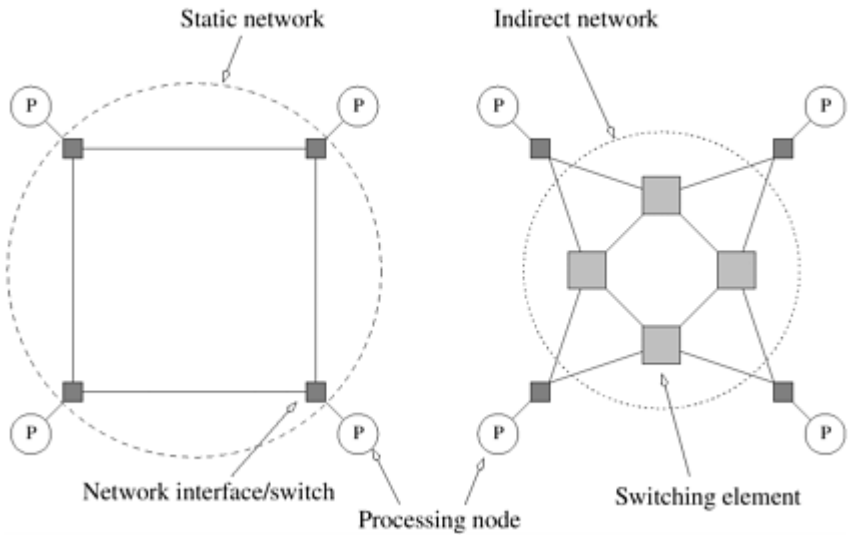
Table 2.1. A summary of the characteristics of various static network topologies connecting p nodes.

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	p - 1	$p(p$ - 1)/2
Star	2	1	1	p - 1
Complete binary tree	2 log((p + 1)/2)	1	1	p - 1
Linear array	p - 1	1	1	p - 1
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	2p
Hypercube	log p	$p/2$	log$p$	(p log p)/2
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	2d	dp

Several metrics described in textbook

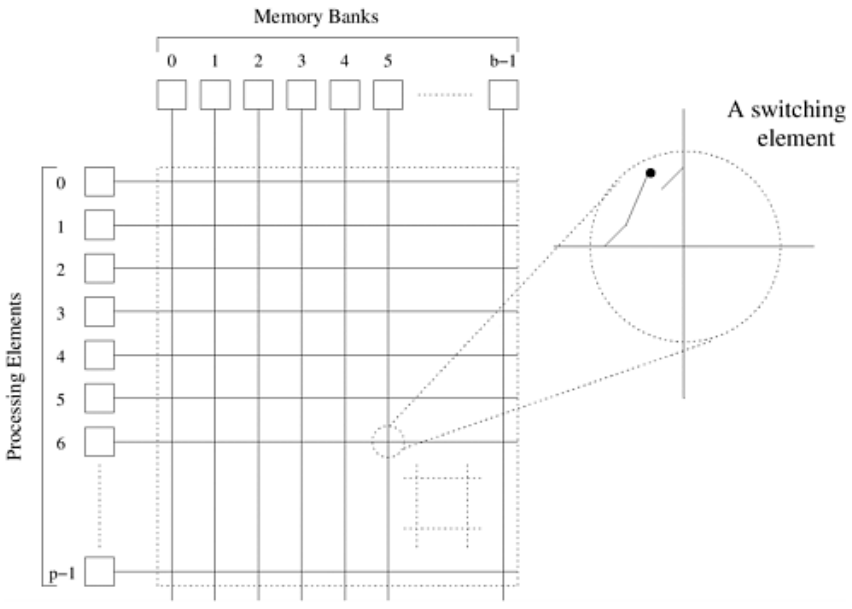
- *Diameter*: how many hops away any two procs can be
- *Bisection width*: number of links to break to partition network
- *Arc Connectivity*: number of paths between two nodes
- *Cost*: can correspond to number of links

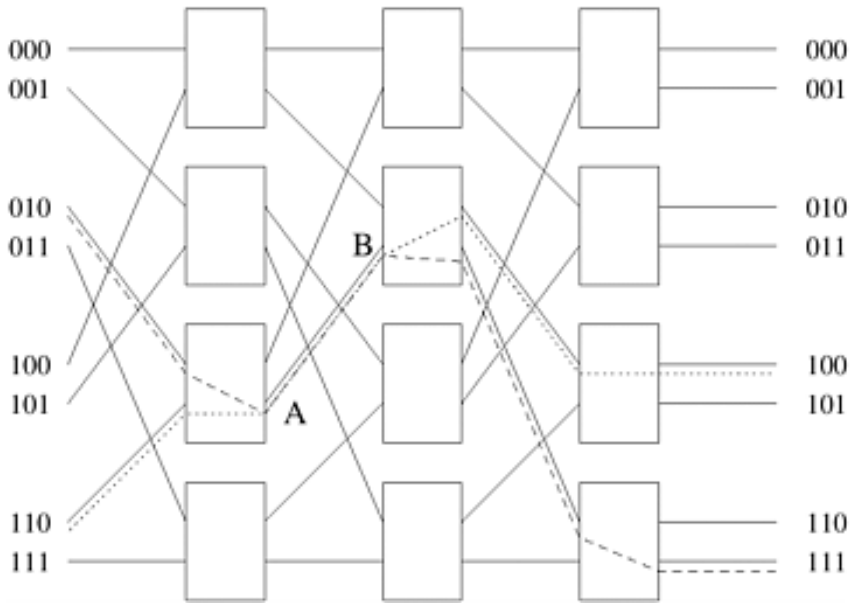
Dynamic Networks



- In a static network, connections are fixed
- Dynamic networks use switches: send data into network with destination, may alter a connection to point in a different direction
- Akin to the internet: packet switching network
- Textbook mixes concepts somewhat: Network for
  - Distributed PEs to communicate
  - PEs to share memory

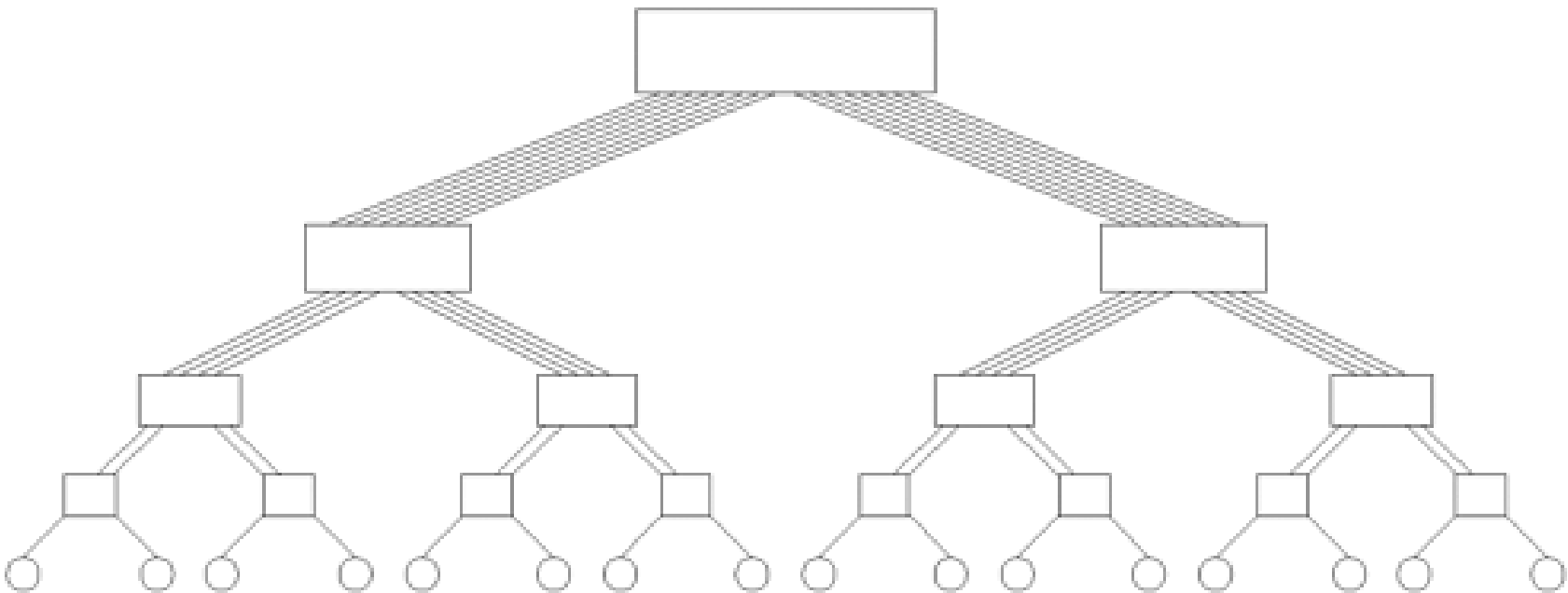
CrossBar and Omega Network





Tree

- Frequently used: Fat tree
- Fairly cost effective: Why?
- What drawbacks might it have?



Routing: Store/Forward Packet, Switching, Cut-Through

- When sending messages, intermediate nodes must decide what to do with a message: Routing protocol/scheme

Store and Forward

- Accumulate the whole message (all M words), store it until it can be forwarded to next hop
- Easy to build but requires large-ish internal buffers and generally has bad performance

Standard Packet Switching

- Break message into chunks (packets)
- Use packet header to carry error-correction info, routing info
- Optimized for the unreliable internet (go around overloaded/dead nodes)
- Better but incurs overhead to solve problems that aren't present in most parallel machines

Routing: Cut-through Routing

- Similar to packet switching: break message into chunks
- Send a *tracer* from source to destination to determine route
- Send message in *flits* (packets) along single route
- Include minimal overhead in packet for error correction, re-routing, etc.
- Cost to communicate message size M between two PEs L hops away

$$t_{comm} = t_s + Lt_h + t_w M$$

The Simplified Model Communication Model

When analyzing performance of programs, consider the following

- t_s: Startup time, incurred once
- t_h: Per-hop time, overhead incurred for each link between source and destination
- t_w: Per-word transfer time between two nodes, takes $t_w \times M$ time for each link between source and destination
- L: number of links to traverse
- M: number of words being sent

Simplified model advocated by Grama et. al

$$t_{comm} = t_s + t_w M$$

- Easy to understand/use
- Relatively easy to apply to programs
- Ignores a pretty big component: why?
- Why would the text adopt this podunk model?

Our Approach

Analyzing Communication Patterns

Will incorporate number of hops L between PEs in the network

$$t_{comm} = t_s + Lt_h + t_w M$$

Try to derive good source/destination pairs and message routes

Analyzing Programs

Will ignore network topology, congestion, number of hops

$$t_{comm} = t_s + t_w M$$

Somewhat unrealistic but makes analysis much simpler