CS252
Graduate Computer Architecture
Lecture 14

Multiprocessor Networks
March 10th, 2010

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

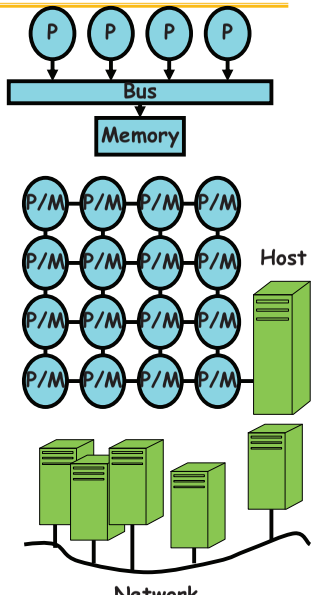
Review: Flynn's Classification (1966)

Broad classification of parallel computing systems

- SISD: Single Instruction, Single Data
  - conventional uniprocessor
- SIMD: Single Instruction, Multiple Data
  - one instruction stream, multiple data paths
  - distributed memory SIMD (MPP, DAP, CM-1&2, Maspar)
  - shared memory SIMD (STARAN, vector computers)
- MIMD: Multiple Instruction, Multiple Data
  - message passing machines (Transputers, nCube, CM-5)
  - non-cache-coherent shared memory machines (BBN Butterfly, T3D)
  - cache-coherent shared memory machines (Sequent, Sun Starfire, SGI Origin)
- MISD: Multiple Instruction, Single Data
  - Not a practical configuration

Review: Examples of MIMD Machines

- Symmetric Multiprocessor
  - Multiple processors in box with shared memory communication
  - Current MultiCore chips like this
  - Every processor runs copy of OS
- Non-uniform shared-memory with separate I/O through host
  - Multiple processors
    - » Each with local memory
    - » general scalable network
  - Extremely light "OS" on node provides simple services
    - » Scheduling/synchronization
  - Network-accessible host for I/O
- Cluster
  - Many independent machine connected with general network
  - Communication through messages



Parallel Programming Models

- Programming model is made up of the languages and libraries that create an abstract view of the machine
- Control
  - How is parallelism created?
  - What orderings exist between operations?
  - How do different threads of control synchronize?
- Data
  - What data is private vs. shared?
  - How is logically shared data accessed or communicated?
- Synchronization
  - What operations can be used to coordinate parallelism
  - What are the atomic (indivisible) operations?
- Cost
  - How do we account for the cost of each of the above?

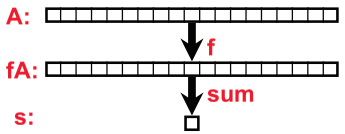
Simple Programming Example

- Consider applying a function f to the elements of an array A and then computing its sum:

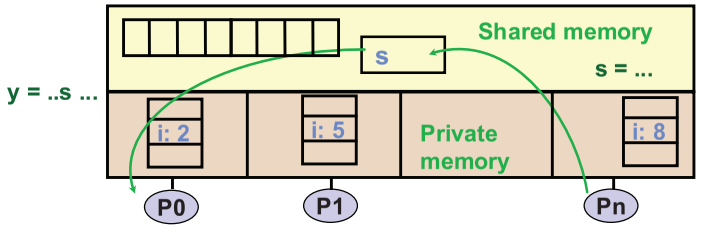
$$\sum_{i=0}^{n-1} f(A[i])$$

- Questions:
  - Where does A live? All in single memory? Partitioned?
  - What work will be done by each processors?
  - They need to coordinate to get a single result, how?

A = array of all data
fA = f(A)
s = sum(fA)



Programming Model 1: Shared Memory



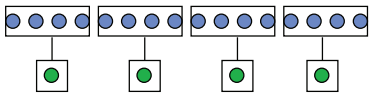
- Program is a collection of threads of control.
  - Can be created dynamically, mid-execution, in some languages
- Each thread has a set of private variables, e.g., local stack variables
- Also a set of shared variables, e.g., static variables, shared common blocks, or global heap.
  - Threads communicate implicitly by writing and reading shared variables.
  - Threads coordinate by synchronizing on shared variables

Simple Programming Example: SM

- Shared memory strategy:
  - small number p << n=size(A) processors
  - attached to single memory

$$\sum_{i=0}^{n-1} f(A[i])$$

- Parallel Decomposition:
  - Each evaluation and each partial sum is a task.
- Assign n/p numbers to each of p procs
  - Each computes independent "private" results and partial sum.
  - Collect the p partial sums and compute a global sum.



Two Classes of Data:

- Logically Shared
  - The original n numbers, the global sum.
- Logically Private
  - The individual function evaluations.
  - What about the individual partial sums?

Shared Memory "Code" for sum

```
static int s = 0;

Thread 1                    Thread 2

for i = 0, n/2-1            for i = n/2, n-1
    s = s + f(A[i])             s = s + f(A[i])
```

- Problem is a race condition on variable s in the program
- A race condition or data race occurs when:
  - two processors (or two threads) access the same variable, and at least one does a write.
  - The accesses are concurrent (not synchronized) so they could happen simultaneously

A Closer Look

A | 3 | 5 | f = square

static int s = 0;

Thread 1		Thread 2	
....		...	
compute f([A[i]) and put in reg0	9	compute f([A[i]) and put in reg0	25
reg1 = s	0	reg1 = s	0
reg1 = reg1 + reg0	9	reg1 = reg1 + reg0	25
s = reg1	9	s = reg1	25
...		...	

- Assume A = [3,5], f is the square function, and s=0 initially
- For this program to work, s should be 34 at the end
  - but it may be 34,9, or 25
- The *atomic* operations are reads and writes
  - Never see ½ of one number, but += operation is not atomic
  - All computations happen in (private) registers

Improved Code for Sum

```
static int s = 0;
static lock lk;
```

```
Thread 1
  local_s1= 0
  for i = 0, n/2-1
    local_s1 = local_s1 + f(A[i])
  lock(lk);
  s = s + local_s1
  unlock(lk);
```

```
Thread 2
  local_s2 = 0
  for i = n/2, n-1
    local_s2= local_s2 + f(A[i])
  lock(lk);
  s = s +local_s2
  unlock(lk);
```

- Since addition is associative, it's OK to rearrange order
- Most computation is on private variables
  - Sharing frequency is also reduced, which might improve speed
  - But there is still a race condition on the update of shared s
- The race condition can be fixed by adding locks (only one thread can hold a lock at a time; others wait for it)

What about Synchronization?

- All shared-memory programs need synchronization
- Barrier – global (/coordinated) synchronization
  - simple use of barriers -- all threads hit the same one

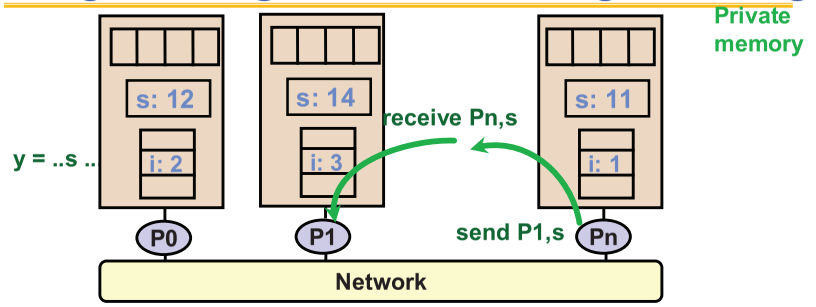
```
work_on_my_subgrid();
barrier;
read_neighboring_values();
barrier;
```

- Mutexes – mutual exclusion locks
  - threads are mostly independent and must access common data

```
lock *l = alloc_and_init();    /* shared */
lock(l);
 access data
unlock(l);
```

- Need atomic operations bigger than loads/stores
  - Actually – Dijkstra's algorithm can get by with only loads/stores, but this is quite complex (and doesn't work under all circumstances)
  - Example: atomic swap, test-and-test-and-set
- Another Option: Transactional memory
  - Hardware equivalent of optimistic concurrency
  - Some think that this is the answer to all parallel programming

Programming Model 2: Message Passing



- Program consists of a collection of named processes.
  - Usually fixed at program startup time
  - Thread of control plus local address space -- NO shared data.
  - Logically shared data is partitioned over local processes.
- Processes communicate by explicit send/receive pairs
  - Coordination is implicit in every communication event.
  - MPI (Message Passing Interface) is the most commonly used SW

Compute A[1]+A[2] on each processor

° First possible solution – what could go wrong?

Processor 1	Processor 2
xlocal = A[1]	xlocal = A[2]
send xlocal, proc2	send xlocal, proc1
receive xremote, proc2	receive xremote, proc1
s = xlocal + xremote	s = xlocal + xremote

° If send/receive acts like the telephone system? The post office?

° Second possible solution

Processor 1	Processor 2
xlocal = A[1]	xloadl = A[2]
send xlocal, proc2	receive xremote, proc1
receive xremote, proc2	send xlocal, proc1
s = xlocal + xremote	s = xlocal + xremote

° What if there are more than 2 processors?

MPI – the de facto standard

- MPI has become the de facto standard for parallel computing using message passing
- Example:

```
for(i=1;i<numprocs;i++) {
  sprintf(buff, "Hello %d! ", i);
  MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG,
           MPI_COMM_WORLD);
}
for(i=1;i<numprocs;i++) {
  MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG,
           MPI_COMM_WORLD, &stat);
  printf("%d: %s\n", myid, buff);
}
```

- Pros and Cons of standards
  - MPI created finally a standard for applications development in the HPC community → portability
  - The MPI standard is a least common denominator building on mid-80s technology, so may discourage innovation

Which is better? SM or MP?

- Which is better, Shared Memory or Message Passing?
  - Depends on the program!
  - Both are "communication Turing complete"
    - » i.e. can build Shared Memory with Message Passing and vice-versa
- Advantages of Shared Memory:
  - Implicit communication (loads/stores)
  - Low overhead when cached
- Disadvantages of Shared Memory:
  - Complex to build in way that scales well
  - Requires synchronization operations
  - Hard to control data placement within caching system
- Advantages of Message Passing
  - Explicit Communication (sending/receiving of messages)
  - Easier to control data placement (no automatic caching)
- Disadvantages of Message Passing
  - Message passing overhead can be quite high
  - More complex to program
  - Introduces question of reception technique (interrupts/polling)

Administrative

- Exam: Next Wednesday (3/17)
  Location: 310 Soda
  TIME: 6:00-9:00
  - This info is on the Lecture page (has been)
  - Get on 8 ½ by 11 sheet of notes (both sides)
  - Meet at LaVal's afterwards for Pizza and Beverages
- I have your proposals.
  We need to meet to discuss them
  - Time this week? Today after class

Paper Discussion: "Future of Wires"

- "Future of Wires," Ron Ho, Kenneth Mai, Mark Horowitz
- Fanout of 4 metric (FO4)
  - FO4 delay metric across technologies roughly constant
  - Treats 8 FO4 as absolute minimum (really says 16 more reasonable)
- Wire delay
  - Unbuffered delay: scales with (length)^2
  - Buffered delay (with repeaters) scales closer to linear with length
- Sources of wire noise
  - Capacitive coupling with other wires: Close wires
  - Inductive coupling with other wires: Can be far wires

"Future of Wires" continued

- Cannot reach across chip in one clock cycle!
  - This problem increases as technology scales
  - Multi-cycle long wires!
- Not really a wire problem – more of a CAD problem??
  - How to manage increased complexity is the issue
- Seems to favor ManyCore chip design??

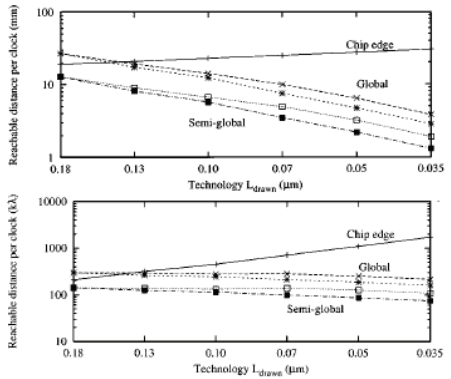


Fig. 20. Reachable distance per clock, repeated global wires.

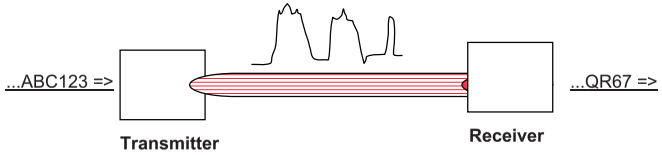
What characterizes a network?

- Topology (what)
  - physical interconnection structure of the network graph
  - direct: node connected to every switch
  - indirect: nodes connected to specific subset of switches
- Routing Algorithm (which)
  - restricts the set of paths that msgs may follow
  - many algorithms with different properties
    - » gridlock avoidance?
- Switching Strategy (how)
  - how data in a msg traverses a route
  - circuit switching vs. packet switching
- Flow Control Mechanism (when)
  - when a msg or portions of it traverse a route
  - what happens when traffic is encountered?

Formalism

- network is a graph V = {switches and nodes} connected by communication channels $C \subseteq V \times V$
- Channel has width w and signaling rate $f = 1/\tau$
  - channel bandwidth $b = wf$
  - phit (physical unit) data transferred per cycle
  - flit - basic unit of flow-control
- Number of input (output) channels is switch *degree*
- Sequence of switches and links followed by a message is a *route*
- Think streets and intersections

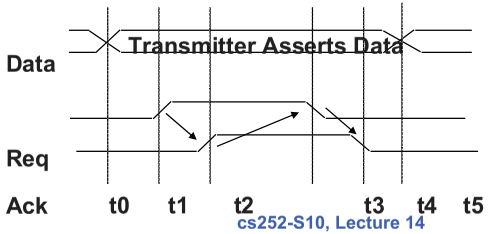
Links and Channels



- *transmitter* converts stream of digital symbols into signal that is driven down the link
- *receiver* converts it back
  - tran/rcv share *physical protocol*
- trans + link + rcv form *Channel* for digital info flow between switches
- *link-level protocol* segments stream of symbols into larger units: packets or messages (framing)
- *node-level protocol* embeds commands for dest communication assist within packet

Clock Synchronization?

- Receiver must be synchronized to transmitter
  - To know when to latch data
- Fully Synchronous
  - Same clock and phase: Isochronous
  - Same clock, different phase: Mesochronous
    - » High-speed serial links work this way
    - » Use of encoding (8B/10B) to ensure sufficient high-frequency component for clock recovery
- Fully Asynchronous
  - No clock: Request/Ack signals
  - Different clock: Need some sort of clock recovery?

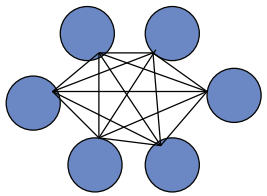


Topological Properties

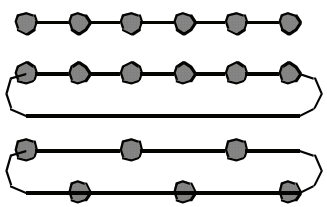
- *Routing Distance* - number of links on route
- *Diameter* - maximum routing distance
- *Average Distance*
- A network is *partitioned* by a set of links if their removal disconnects the graph

Interconnection Topologies

- Class of networks scaling with N
- Logical Properties:
  - distance, degree
- Physical properties
  - length, width
- Fully connected network
  - diameter = 1
  - degree = N
  - cost?
    - » bus => O(N), but BW is O(1) - actually worse
    - » crossbar => $O(N^2)$ for BW O(N)
- VLSI technology determines switch degree



Example: Linear Arrays and Rings



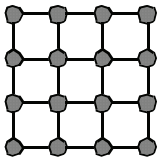
Linear Array

Torus

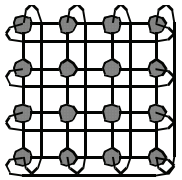
Torus arranged to use short wires

- Linear Array
  - Diameter?
  - Average Distance?
  - Bisection bandwidth?
  - Route A -> B given by relative address R = B-A
- Torus?
- Examples: FDDI, SCI, FiberChannel Arbitrated Loop, KSR1

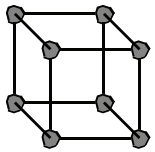
Example: Multidimensional Meshes and Tori



2D Grid



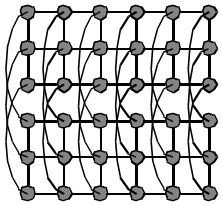
2D Torus



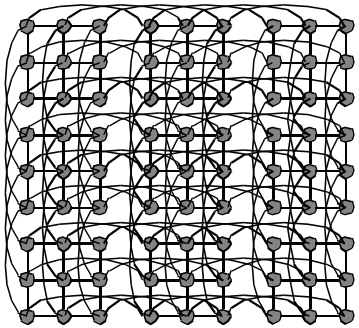
3D Cube

- *n*-dimensional array
  - N = k_{n-1} X ...X k_0 nodes
  - described by *n*-vector of coordinates (i_{n-1}, ..., i_0)
- *n*-dimensional *k*-ary mesh: N = k^n
  - k = $\sqrt[n]{N}$
  - described by *n*-vector of radix k coordinate
- *n*-dimensional *k*-ary torus (or *k*-ary *n*-cube)?

On Chip: Embeddings in two dimensions

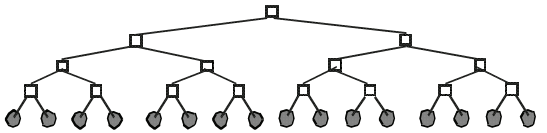


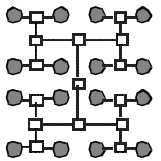
6 x 3 x 2



- Embed multiple logical dimension in one physical dimension using long wires
- When embedding higher-dimension in lower one, either some wires longer than others, or all wires long

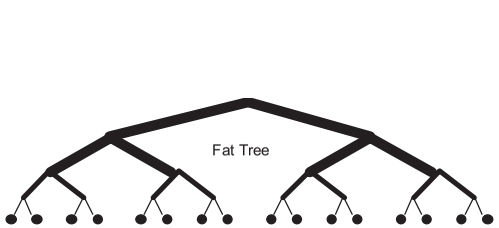
Trees

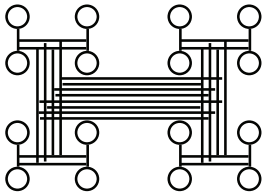




- Diameter and ave distance logarithmic
  - k-ary tree, height n = $\log_k$ N
  - address specified n-vector of radix k coordinates describing path down from root
- Fixed degree
- Route up to common ancestor and down
  - R = B xor A
  - let i be position of most significant 1 in R, route up i+1 levels
  - down in direction given by low i+1 bits of B
- H-tree space is O(N) with O(√N) long wires
- Bisection BW?

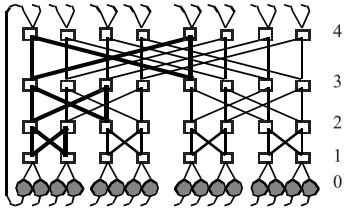
Fat-Trees



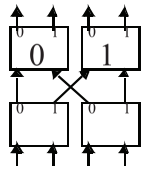


- Fatter links (really more of them) as you go up, so bisection BW scales with N

Butterflies



16 node butterfly



building block

- Tree with lots of roots!
- N log N (actually N/2 x logN)
- Exactly one route from any source to any dest
- R = A xor B, at level i use 'straight' edge if r_i=0, otherwise cross edge
- Bisection N/2 vs N $^{(n-1)/n}$ (for n-cube)

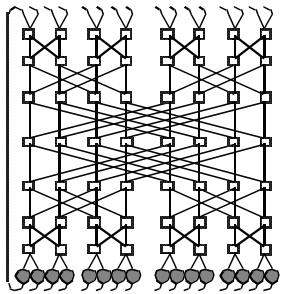
k-ary n-cubes vs k-ary n-flies

- degree n vs degree k
- N switches vs N log N switches
- diminishing BW per node vs constant
- requires locality vs little benefit to locality

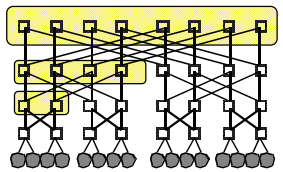
- Can you route all permutations?

Benes network and Fat Tree

16-node Benes Network (Unidirectional)



16-node 2-ary Fat-Tree (Bidirectional)

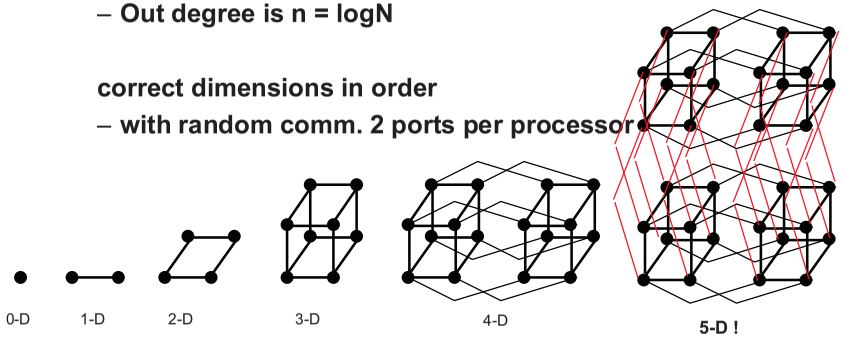


- Back-to-back butterfly can route all permutations
- What if you just pick a random mid point?

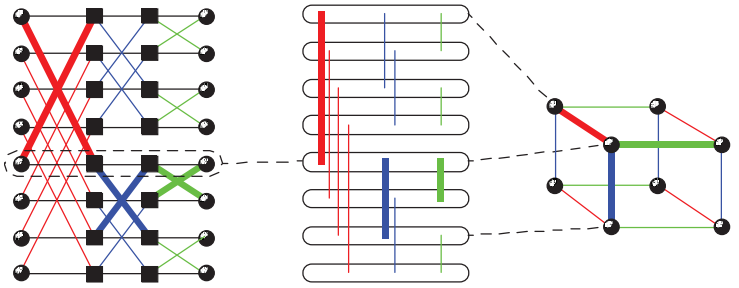
Hypercubes

- Also called binary n-cubes. # of nodes = N = 2^n.
- O(logN) Hops
- Good bisection BW
- Complexity
  - Out degree is n = logN

  correct dimensions in order
  - with random comm. 2 ports per processor



Relationship BttrFlies to Hypercubes



- Wiring is isomorphic
- Except that Butterfly always takes log n steps

Real Machines

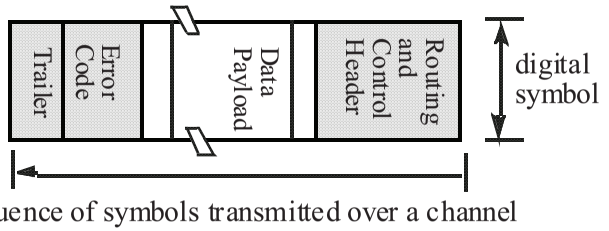
Machine	Topology	Cycle Time (ns)	Channel Width (bits)	Routing Delay (cycles)	Flit (data bits)
nCUBE/2	Hypercube	25	1	40	32
TMC CM-5	Fat-Tree	25	4	10	4
IBM SP-2	Banyan	25	8	5	16
Intel Paragon	2D Mesh	11.5	16	2	16
Meiko CS-2	Fat-Tree	20	8	7	8
CRAY T3D	3D Torus	6.67	16	2	16
DASH	Torus	30	16	2	16
J-Machine	3D Mesh	31	8	2	8
Monsoon	Butterfly	20	16	2	16
SGI Origin	Hypercube	2.5	20	16	160
Myricom	Arbitrary	6.25	16	50	16

- Wide links, smaller routing delay
- Tremendous variation

Some Properties

- Routing
  - relative distance: R = ($b_{n-1} - a_{n-1}$, ... , $b_0 - a_0$)
  - traverse ri = $b_i - a_i$ hops in each dimension
  - *dimension-order routing? Adaptive routing?*
- Average Distance — Wire Length?
  - n x 2k/3 for mesh
  - nk/2 for cube
- Degree?
- Bisection bandwidth? — Partitioning?
  - k^{n-1} bidirectional links
- Physical layout?
  - 2D in O(N) space — Short wires
  - higher dimension?

Typical Packet Format

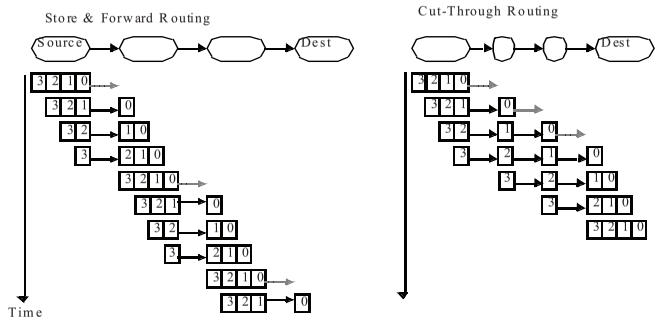


- **Two basic mechanisms for abstraction**
  - **encapsulation**
  - **Fragmentation**
- **Unfragmented packet size S = S_{data}+$S_{encapsulation}$**

Communication Perf: Latency per hop

- Time$(S)_{s-d}$ = overhead + routing delay + channel occupancy + contention delay
- Channel occupancy = S/b = (S_{data}+ $S_{encapsulation}$)/b
- Routing delay?
- Contention?

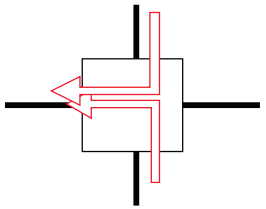
Store&Forward vs Cut-Through Routing



Time:	**h(S/b + Δ/τ)**	**vs**	**S/b + h Δ/τ**
OR(cycles):	**h(S/w + Δ)**	**vs**	**S/w + h Δ**

- **what if message is fragmented?**
- **wormhole vs virtual cut-through**

Contention

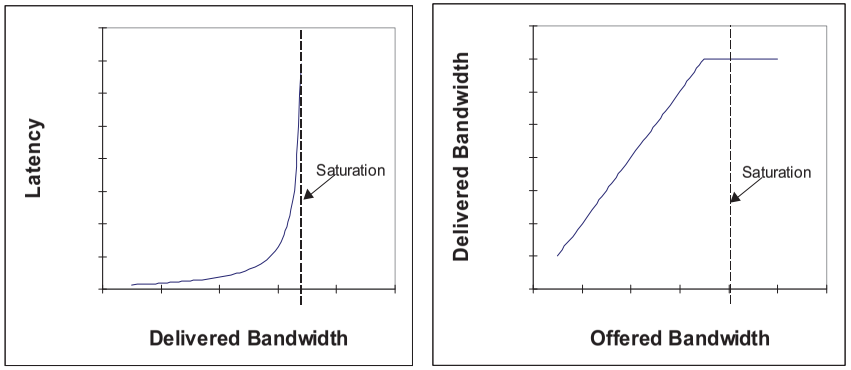


- **Two packets trying to use the same link at same time**
  - **limited buffering**
  - **drop?**
- **Most parallel mach. networks block in place**
  - **link-level flow control**
  - **tree saturation**
- **Closed system - offered load depends on delivered**
  - **Source Squelching**

Bandwidth

- **What affects local bandwidth?**
  - **packet density** **b x S_{data}/n**
  - **routing delay** **b x S_{data} /(n + wΔ)**
  - contention
    - » endpoints
    - » within the network
- Aggregate bandwidth
  - bisection bandwidth
    - » sum of bandwidth of smallest set of links that partition the network
  - total bandwidth of all the channels: Cb
  - suppose N hosts issue packet every M cycles with ave dist
    - » each msg occupies h channels for ℓ = n/w cycles each
    - » C/N channels available per node
    - » link utilization for store-and-forward: $\rho = (h\ell/M$ channel cycles/node)/(C/N) = $Nh\ell/MC < 1$!
    - » link utilization for wormhole routing?

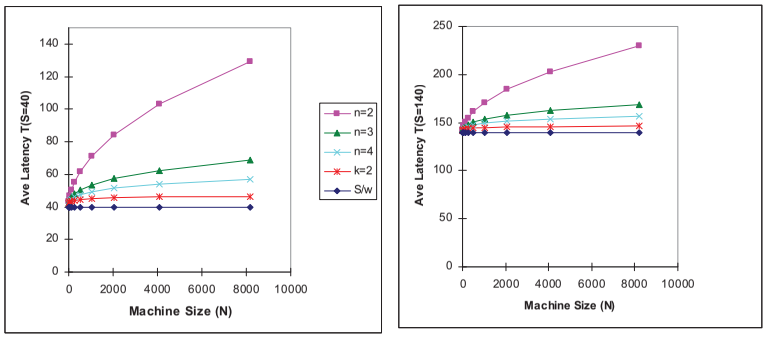
Saturation



How Many Dimensions?

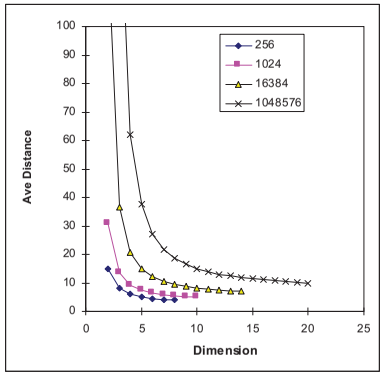
- **n = 2 or n = 3**
  - **Short wires, easy to build**
  - **Many hops, low bisection bandwidth**
  - **Requires traffic locality**
- **n >= 4**
  - **Harder to build, more wires, longer average length**
  - **Fewer hops, better bisection bandwidth**
  - **Can handle non-local traffic**
- **k-ary n-cubes provide a consistent framework for comparison**
  - **N = k^n**
  - **scale dimension (n) or nodes per dimension (k)**
  - **assume cut-through**

Traditional Scaling: Latency scaling with N



- **Assumes equal channel width**
  - **independent of node count or dimension**
  - **dominated by average distance**

Average Distance

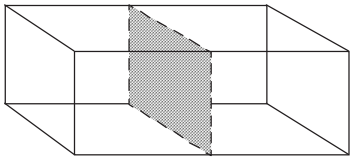


ave dist = n(k-1)/2

- but, equal channel width is not equal cost!
- Higher dimension => more channels

Dally Paper: In the 3D world

- For N nodes, bisection area is $O(N^{2/3})$



- For large N, bisection bandwidth is limited to $O(N^{2/3})$
  - Bill Dally, IEEE TPDS, [Dal90a]
  - For fixed bisection bandwidth, low-dimensional k-ary n-cubes are better (otherwise higher is better)
  - i.e., a few short fat wires are better than many long thin wires
  - What about many long fat wires?

Dally paper (con't)

- Equal Bisection,W=1 for hypercube ⇒ W= ½k
- Three wire models:
  - Constant delay, independent of length
  - Logarithmic delay with length (exponential driver tree)
  - Linear delay (speed of light/optimal repeaters)

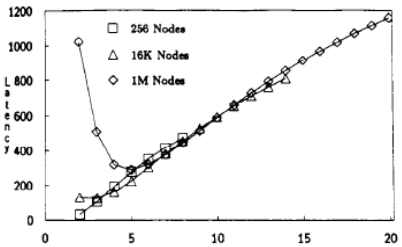


Fig. 9. Latency versus dimension for 256, 16K, and 1M nodes, logarithmic delay.

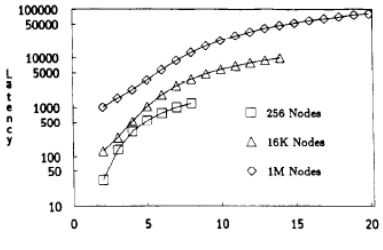


Fig. 10. Latency versus dimension for 256, 16K, and 1M nodes, linear delay.

Logarithmic Delay

Linear Delay

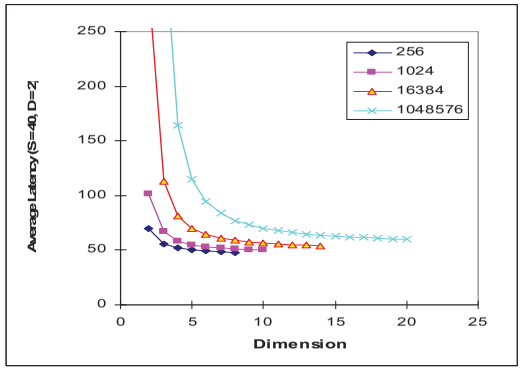
Equal cost in k-ary n-cubes

- Equal number of nodes?
- Equal number of pins/wires?
- Equal bisection bandwidth?
- Equal area?
- Equal wire length?

What do we know?

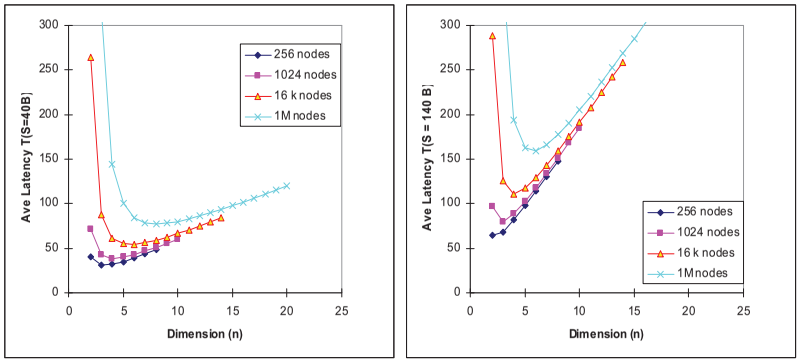
- switch degree: n diameter = n(k-1)
- total links = Nn
- pins per node = 2wn
- bisection = k^{n-1} = N/k links in each directions
- 2Nw/k wires cross the middle

Latency for Equal Width Channels



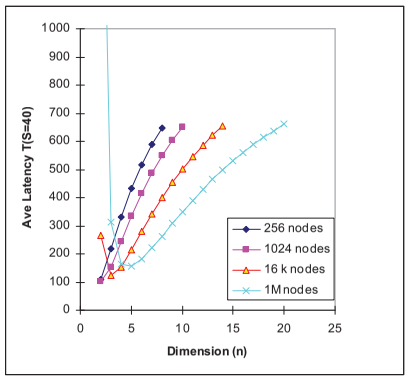
- total links(N) = Nn

Latency with Equal Pin Count



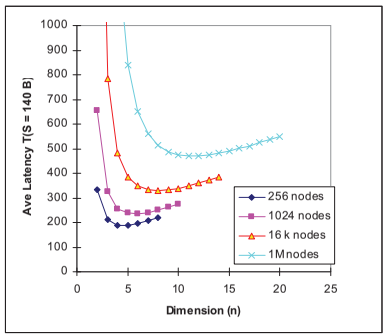
- Baseline n=2, has w = 32 (128 wires per node)
- fix 2nw pins => w(n) = 64/n
- distance up with n, but channel time down

Latency with Equal Bisection Width



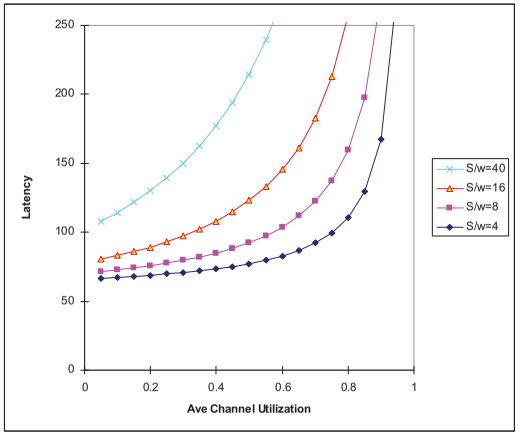
- N-node hypercube has N bisection links
- 2d torus has $2N^{1/2}$
- Fixed bisection $\Rightarrow$ $w(n) = N^{1/n} / 2 = k/2$
- 1 M nodes, n=2 has w=512!

Larger Routing Delay (w/ equal pin)



- Dally's conclusions strongly influenced by assumption of small routing delay
  - Here, Routing delay Δ=20

Saturation



- Fatter links shorten queuing delays

Discussion

- Rich set of topological alternatives with deep relationships
- Design point depends heavily on cost model
  - nodes, pins, area, ...
  - Wire length or wire delay metrics favor small dimension
  - Long (pipelined) links increase optimal dimension
- Need a consistent framework and analysis to separate opinion from design
- Optimal point changes with technology

Summary

- Programming Models:
  - Shared Memory
  - Message Passing
- Networking and Communication Interfaces
  - Fundamental aspect of multiprocessing
- Network Topologies:

Topology	Degree	Diameter	Ave Dist	Bisection	D (D ave) @ P=1024
1D Array	2	N-1	N / 3	1	huge
1D Ring	2	N/2	N/4	2	
2D Mesh	4	$2 (N^{1/2} - 1)$	$2/3\ N^{1/2}$	$N^{1/2}$	63 (21)
2D Torus	4	$N^{1/2}$	$1/2\ N^{1/2}$	$2N^{1/2}$	32 (16)
k-ary n-cube	2n	nk/2	nk/4	nk/4	15 (7.5) @n=3
Hypercube	n =log N		n	n/2	N/2 10 (5)

- Fair metrics of comparison
  - Equal cost: area, bisection bandwidth, etc