Massively Parallel Processors

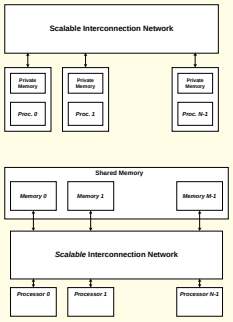
ECE/CS 757 Spring 2007
J. E. Smith

Lecture Outline

- Introduction
- Software Scaling
- Hardware Scaling
- Case studies
  - MIT J Machine
  - Cray T3D
  - Cray T3E*
  - CM-5**
- Readings (to be discussed on Friday, 4/24)
  - *) Steven L. Scott, Synchronization and Communication in the T3E Multiprocessor, Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pages 26-36, October 1996.
  - **) W. Daniel Hillis, Lewis W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer," CACM, pp. 30-40, Nov. 1993.

MPP Definition etc.

- A (large) bunch of computers connected with a (very) high performance network
  - Primarily execute highly parallel application programs
- Applications
  - Typically number crunching
  - Also used for computation-intensive commercial apps
    - e.g. data mining
- May use distributed memory
  - Computers or small SMP as nodes of large distributed memory system
- OR shared memory
  - Processors connected to large shared memory
    - Less common today
- Also hybrids
  - Shared real space, assists for load/stores



Scalability

- Term comes up often in MPP systems
- Over time:
  - Computer system components become smaller and cheaper
    ⇒ more processors, more memory
  - Range of system sizes within a product family
  - Problem sizes become larger
    simulate the entire airplane rather than the wing
  - Required accuracy becomes greater
    forecast the weather a week in advance rather than 3 days
- Should designers come up with new system architectures for each generation?
  - Or design a scalable architecture that can survive for many generations
  - And be useful for a range of systems within a product family

Scaling

- How do algorithms and hardware behave as systems, size, accuracies become greater?
- Intuitively: "Performance" should scale linearly with cost
  - But, easier said than done
- Software Scaling
  - Algorithms, problem size, computational complexity, error analysis
- Hardware Scaling
  - Lower level performance features "scaling" together

Cost

- Cost is a function of more than just the processor.
  - Memory
  - Interconnect
  - I/O
- Cost is a complex function of many hardware components and software
- Cost is often not a "smooth" function
  - Often a function of packaging
    how many pins on a board
    how many processors on a board
    how many boards in a chassis

Performance

- How does performance vary with added processors?
  - Depends on inherently serial portion vs. parallel portion
  - Depends on problem size
  - Depends on architecture and algorithm
  - Depends on computation vs. communication

Speedup Review

Let Speedup = T_{serial} / $T_{parallel}$

- Amdahl's law

  f = fraction of serial work;

  (1-f) = parallel fraction

- Speedup with N processors, S(N) = 1 / (f + (1-f)/N))

  Maximum speedup = 1/f

  Eg. 10% serial work => maximum speedup is 10.

Effect of Problem Size

- Amdahl's law assumes constant problem size
  - Or, serial portion grows linearly with parallel portion
- Often, serial portion does *not* grow linearly with parallel portions
  - And, parallel processors solve larger problems.
- Example: NxN Matrix multiplication

  Initialize matrices, serial, complexity N

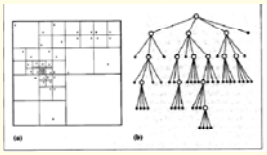
  Multiply matrices, parallel, complexity N^3

Problem Constrained Scaling

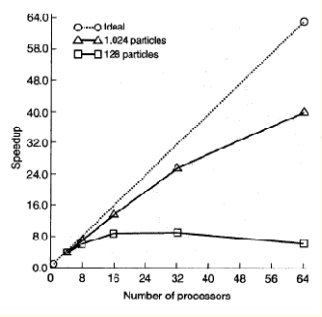
- User wants to solve same problem, only faster
  - E.g., Video compression
- Amdahl's law is a limitation
- In many cases, problem sizes grow

Example: Barnes-Hut Galaxy Simulation

- Simulates gravitational interactions of N-bodies in space
  - N^2 complexity
- Partition space into regions with roughly equal numbers of bodies
  - Model region as a single point w/ gravity at center
  - Becomes NlogN complexity



Galaxy Simulation w/ Constant Problem Scaling



Memory Constrained Scaling

- **Let problem size scale linearly with number of processors.**
  (assumes memory scales linearly with no. of processors)
- **Scaled Speedup: rate(p)/rate(1)**

  $Speedup_{MC}(p) = work(p)/time(p)*time(1)/work(1)$

- **Even with good speedups, can lead to large increases in execution time if work grows faster than linearly in memory usage**

Memory Constrained Scaling

- **Matrix multiply example:**
  f = N / (N + N^3), and N grows so that N^3 term dominates
  S(1) = 1
  S(10) ~ 10
  S(100) ~ 100
  S(1000) ~ 1000
- **BUT, 1000 times increase in problem size ⇒ 1,000,000 times increase in execution time, even with 1000 processors.**

Time Constrained Scaling

- **Execution time is kept fixed as system scales**
  - User has fixed time to use machine or wait for result
  - Often realistic (e.g., best weather forecast overnight)
- **Performance = Work/Time and time is constant, so:**

  $SpeedupTC(p) = work(p)/work(1)$

Time Constrained Scheduling

- **Overheads can become a problem:**
- **For matrix multiply, data set size can be increased by $N^{1/3}$**
  ⇒ for 1000 x more processors, data size increases by 10.
- **Problem grows slower than processors,**
- **Eventually performance gain will flatten**
  - And diminish due to overheads associated with smaller amounts of data per processor.
  - Start with 100 element array ⇒ 100 elements per 1 processor
  - Scale up to 1000 processors ⇒ 1 element per processor

Scaling Methodology

- **Often problem sizes are increased to reduce error**
  - E.g. finer grids or time steps
- **Must understand application to scale meaningfully (would user scale grid, time step, error bound, or some combination?)**
- **Equal Error Scaling**
  - Scale problem so that all sources of error have equal contribution to total error

Example: Barnes-Hut Galaxy Simulation

- **Different parameters govern different sources of error**
  - Number of bodies (*n*)
  - Time-step resolution (dt)
  - Force calculation accuracy (fa)
- **Scaling Rule**
  All components of simulation error should scale at the same rate
- **Naïve memory constrained scaling**
  - Scaling up problem size (and number of processors)
  - Increases total execution time slightly (due to nlogn nature of problem
- **Equal error scaling**
  - Scaling up by a factor of k adds an additional factor of $k^{3/4}$

Example: Barnes-Hut Galaxy Simulation

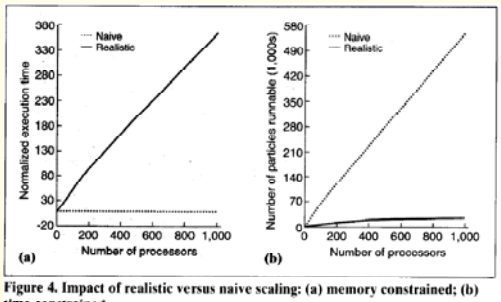


Figure 4. Impact of realistic versus naive scaling: (a) memory constrained; (b) time constrained.

Hardware Scaling

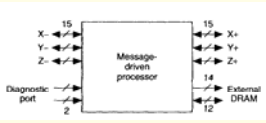
- **Bandwidth Scaling**
  - should increase proportional to # procs.
  - crossbars
  - multi-stage nets
- **Latency Scaling**
  - ideally remain constant
  - in practice, logn scaling can be achieved
  - local memories help (local latency may be more important than global)
  - latency may be dominated by overheads, anyway
- **Cost Scaling**
  - low overhead cost (most cost incremental)
  - in contrast to many large SMPs
- **Physical Scaling**
  - loose vs. dense packing
  - chip level integration vs. commodity parts

Case Study: MIT J-Machine

- **Each node a small message-driven processor**
- **HW support to queue msgs and dispatch to msg handler task**

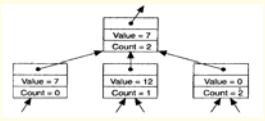
J-Machine Message-Driven Processor

- **MIT research project**
- **Targets fine-grain message passing**
  - very low message overheads allow:
    - small messages
    - small tasks
- **J-Machine architecture**
  - 3D mesh direct interconnect
  - Global address space
  - up to 1 Mwords of DRAM per node
  - MDP single-chip supports
    - processing
    - memory control
    - message passing
  - *not* an off-the-shelf chip



Features/Example: Combining Tree

- **Each node collects data from lower levels, accumulates sum, and passes sum up tree when all inputs are done.**
- **Communication**
  - SEND instructions send values up tree in small messages
  - On arrival, a task is created to perform COMBINE routine
- **Synchronization**
  - message arrival dispatches a task
  - example: combine message invokes COMBINE routine
  - presence bits (full/empty) on state
    - value set empty; reads are blocked until it becomes full



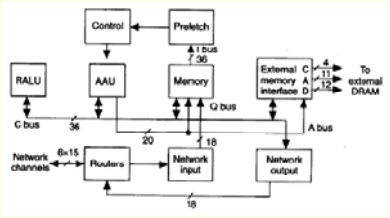
Example: Combining Tree

- **Code for Call Message:**

```
COMBINE:  MOVE    [1,A3], COMB          ; get node pointer from msg
          MOVE    [2,A3], R1            ; get value from msg
          ADD     R1, COMB.VALUE, R1
          MOVE    R1, COMB.VALUE        ; store result
          MOVE    COMB.COUNT, R2        ; get Count
          ADD     R2, -1, R2
          MOVE    R2, COMB.COUNT        ; store decremented Count
          BNZ     R2, DONE
          MOVE    HEADER.R0             ; get message header
          SEND2   COMB.PARENT_NODE, R0  ; send message to parent
          SEND2E  COMB.PARENT, R1       ; with value
DONE:     SUSPEND
```

Block Diagram

- Processor + Communication

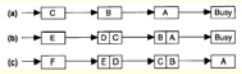


Network Architecture

- 3D Mesh; up to 64K nodes
- No torus ⇒ faces for I/O
- Bidirectional channels
  - channels can be turned around on alternate cycles
  - 9 bits data + 6 bits control
  - ⇒ 9 bit phit
  - 2 phit per flit (granularity of flow control)
  - Each channel 288 Mbps
  - Bisection bandwidth (1024 nodes) 18.4 Gps
- Synchronous routers
  - clocked at 2X processor clock
  - ⇒ 9-bit phit per 62.5ns
  - messages route at 1 cycle per hop

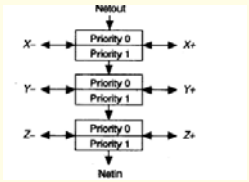
Network Architecture: Flow Control

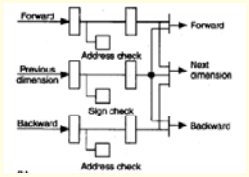
- 2 Phits per Flit
- Compression x2 on blockage



Router

- Three independent routers; one per dimension
  - Dimension order routing
- Two priorities (virtual networks) per dimension





Messages

Table 1. A typical message in the J-Machine.

Flit	Contents	Remark
1	5:+	*X* address
2	1:−	*Y* address
3	4:+	*Z* address
4	MSG: 00	Method to call
5	00440	
6	INT: 00	Argument to method
7	0023	
8	INT: 00	Reply address
9	<1:5:2> T	

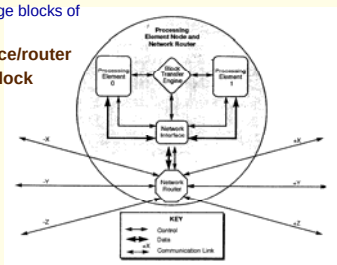
The first three flits contain the destination address. The final flit in the message is marked as the tail.

Case Study: Cray T3D

- Processing element nodes
- 3D Torus interconnect
- Wormhole routing
- PE numbering
- Local memory
- Support circuitry
- Prefetch
- Messaging
- Barrier synch
- Fetch and inc.
- Block transfers

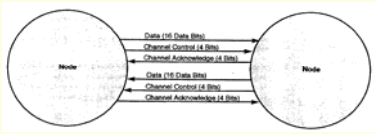
Processing Element Nodes

- Two Alpha 21064 processors per node
- Shared block transfer engine (BLT)
  - DMA-like transfer of large blocks of data
- Shared network interface/router
- Synchronous 6.67 ns clock



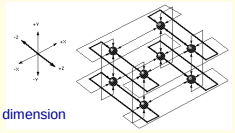
Communication Links

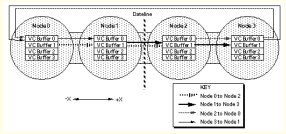
- Signals:
  - Data: 16 bits
  - Channel control: 4 bits
  - -- request/response, virt. channel buffer
  - Channel acknowledge: 4 bits
- virt. channel buffer status



Routing

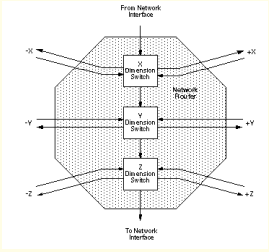
- 3D Torus
- Dimension order routing
  - may go in either + or - direction along a dimension
- Virtual channels
  - Four virtual channel buffers per physical channel
    ⇒ two request channels, two response channels
- Deadlock avoidance
  - In each dimension specify a "dateline" link
  - Packets that do not cross dateline use virtual channel 0
  - Packets that cross dateline use virtual channel 1





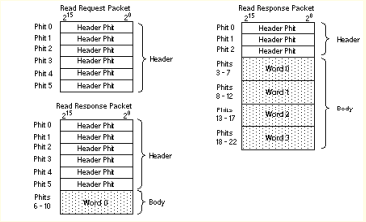
Network Routers

- Separate X,Y,Z dimensions



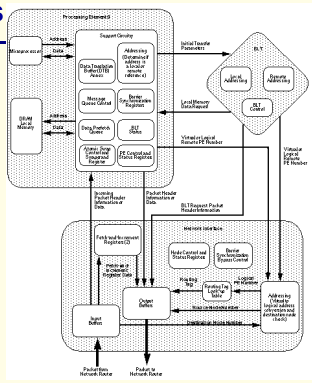
Packets

- Size: 8 physical units (phits)
  - 16 bits per phit
- Header:
  - routing info
  - destn processor
  - control info
  - source processor
  - memory address
- Read Request
  - header: 6 phits
- Read Response
  - header: 6 phits
  - body: 4 phits (1 word)
  - or 16 phits (4 words)



Processing Nodes

- Processor: Alpha 21064
- Support circuitry:
  - Address interpretation
  - reads and writes (local/non-local)
  - data prefetch
  - messaging
  - barrier synch.
  - fetch and incr.
  - status



Processing Element Numbering

- **Physical**
  - Includes all PEs in system
- **Logical**
  - Ignores spare PEs; allows spares for failed nodes
  - These are available to software
- **Virtual**
  - What the application software sees
  - OS does virtual to logical mapping

Address Interpretation

- **T3D needs:**
  - 64 MB memory per node => 26 bits
  - up to 2048 nodes => 11 bits
- **Alpha 21064 provides:**
  - 43-bit virtual address
  - 32-bit physical address
    (+2 bits for mem mapped devices)

  ⇒ Annex registers in DTB
  - external to Alpha
  - 32-annex registers
  - map 32-bit address onto 48 bit node + 27-bit address
  - annex registers also contain function info
  - e.g. cache / non-cache accesses
  - DTB modified via load/locked store cond. insts.

Data Prefetch

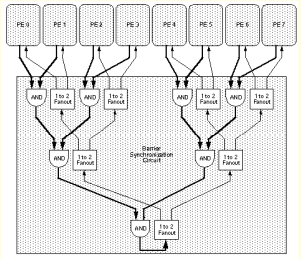
- **Architected Prefetch Queue**
  - 1 word wide by 16 deep
- **Prefetch instruction:**
  - Alpha prefetch hint => T3D prefetch
- **Performance**
  - Allows multiple outstanding read requests
  - (normal 21064 reads are blocking)

Messaging

- **Message queues**
  - 256 KBytes reserved space in local memory
  - => 4080 message packets + 16 overflow locations
- **Sending processor:**
  - Uses Alpha PAL code
  - builds message packets of 4 words
  - plus address of receiving node
- **Receiving node**
  - stores message
  - interrupts processor
  - processor sends an ack
  - processor may execute routine at address provided by message
  - (active messages)
  - if message queue full; NACK is sent
  - also, error messages may be generated by support circuitry

Barrier Synchronization

- **For Barrier or Eureka**
- **Hardware implementation**
  - hierarchical tree
  - bypasses in tree to limit its scope
  - masks for barrier bits to further limit scope
  - interrupting or non-interrupting



Fetch and Increment

- **Special hardware registers**
  - 2 per node
  - user accessible
  - used for auto-scheduling tasks
  - (often loop iterations)

Block Transfer

- Special block transfer engine
  - does DMA transfers
  - can gather/scatter among processing elements
  - up to 64K packets
  - 1 or 4 words per packet
- Types of transfers
  - constant stride read/write
  - gather/scatter
- Requires System Call
  - for memory protection
    => big overhead

Cray T3E

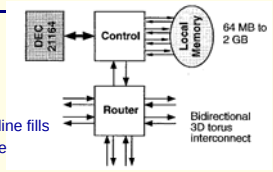
- T3D Post Mortem
- T3E Overview
- Global Communication
- Synchronization
- Message passing
- Kernel performance

T3D Post Mortem

- Very high performance interconnect
  - 3D torus worthwhile
- Barrier network "overengineered"
  - Barrier synch not a significant fraction of runtime
- Prefetch queue useful; should be more of them
- Block Transfer engine not very useful
  - high overhead to setup
  - yet another way to access memory
- DTB Annex difficult to use in general
  - one entry might have sufficed
  - every processor must have same mapping for physical page

T3E Overview

- Alpha 21164 processors
- Up to 2 GB per node
- Caches
  - 8K L1 and 96K L2 on-chip
  - supports 2 outstanding 64-byte line fills
  - stream buffers to enhance cache
  - only local memory is cached
  - => hardware cache coherence straightforward
- 512 (user) + 128 (system) E-registers for communication/synchronization
- One router chip per processor



T3E Overview, contd.

- Clock:
  - system (i.e. shell) logic at 75 MHz
  - proc at some multiple (initially 300 MHz)
- 3D Torus Interconnect
  - bidirectional links
  - adaptive multiple path routing
  - links run at 300 MHz

Global Communication: E-registers

- General Issue:
  - Cache line-based microprocessor interface inadequate
    For strided accesses
    For multiple outstanding accesses
  - Also, limited physical address space
- Extend address space
- Implement "centrifuging" function
- Memory-mapped (in IO space)
- Operations:
  - load/stores between E-registers and processor registers
  - Global E-register operations
    transfer data to/from global memory
    messaging
    synchronization

Global Address Translation

- **E-reg block holds base and mask;**
  - previously stored there as part of setup
- **Remote memory access (mem mapped store):**
  - data bits: E-reg pointer(8) + address index(50)
  - address bits: Command + src/dstn E-reg

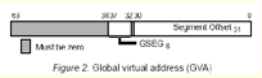
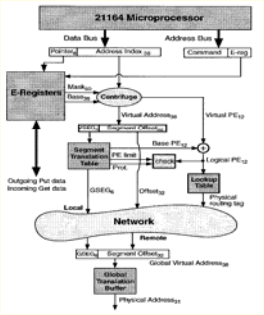


Figure 2. Global virtual address (GVA)

Global Address Translation



Address Translation, contd.

- **Translation steps**
  - Address index centrifuged with mask => virt address + virt PE
  - Offset added to base => vseg + seg offset
  - vseg translated => gseg + base PE
  - base PE + virtual PE => logical PE
  - logical PE through lookup table => physical routing tag
- **GVA: gseg(6) + offset (32)**
  - goes out over network to physical PE
    at remote node, global translation => physical address

Global Communication: Gets and Puts

- **Get: global read**
  - word (32 or 64-bits)
  - vector (8 words, with stride)
  - stride in access E-reg block
- **Put: global store**
- **Full/Empty synchronization on E-regs**
- **Gets/Puts may be pipelined**
  - up to 480 MB/sec transfer rate between nodes

Synchronization

- **Atomic ops between E-regs and memory**
  - Fetch & Inc
  - Fetch & Add
  - Compare & Swap
  - Masked Swap
- **Barrier/Eureka Synchronization**
  - 32 BSUs per processor
  - accessible as memory-mapped registers
    protected via address translation
  - BSUs have states and operations
  - State transition diagrams
  - Barrier/Eureka trees are embedded in 3D torus
  - use highest priority virtual channels

State	Description
S_EUR	A eureka event came
S_EUR_I	A eureka came, interrupt signalled
S_ARM	Barrier is armed
S_ARM_I	Barrier is armed, an interrupt will occur on completion
S_BAR	Barrier just completed
S_BAR_I	Barrier just completed, interrupt signalled

Table 2. Barrier/Eureka Synchronization Unit States

Operation	Description
OP_EUR	Send eureka
OP_INT	Set to interrupt when a eureka event occurs
OP_BAR	Arm Barrier
OP_BAR_I	Arm Barrier, interrupt on completion
OP_EUR_B	Send eureka and arm barrier

Table 3. Barrier/Eureka Synchronization Unit Operations

Message Passing

- **Message queues**
  - arbitrary number
  - created in user or system mode
  - mapped into memory space
  - up to 128 MBytes
- **Message Queue Control Word in memory**
  - tail pointer
  - limit value
  - threshold triggers interrupt
  - signal bit set when interrupt is triggered
- **Message Send**
  - from block of 8 E-regs
  - Send command, similar to put
- **Queue head management done in software**
  - swap can manage queue in two segments

Figure 5. Message Queue Control Word

Performance: Pipelined Memory Access

- Load an a 128KB array from a node three hops away
- Vary number of E registers

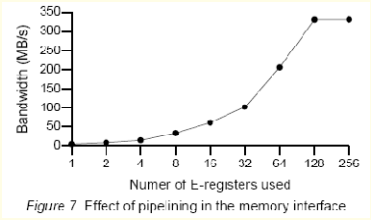


Figure 7. Effect of pipelining in the memory interface

Performance: Startup Latency

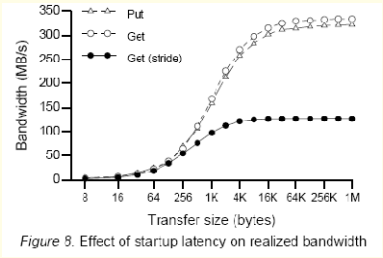


Figure 9. Effect of startup latency on realized bandwidth

Messaging Performance

- Procs 1-15 share messages w/ proc 0
- At peak, proc. 0 takes 1.07 usec to receive message and reply

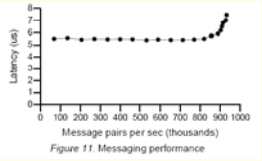


Figure 11. Messaging performance

Barrier Performance

- Hardware vs. Software

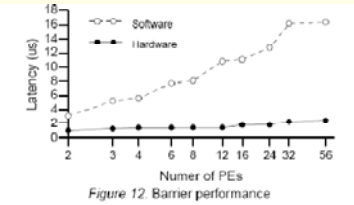


Figure 12. Barrier performance

T3E Summary

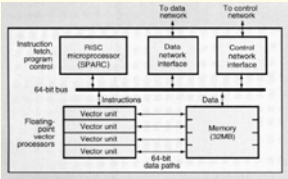
- Messaging is done well...
  - within constraints of COTS processor
- Relies more on high communication and memory bandwidth than caching
  - => lower perf on dynamic irregular codes
  - => higher perf on memory-intensive codes with large communication
- Centrifuge: probably unique
- Barrier synch uses dedicated hardware but NOT dedicated network

Case Study: Thinking Machines CM5

- Follow-on to CM2
  - Abandons SIMD, bit-serial processing
  - Uses off-shelf processors/parts
  - Focus on floating point
  - 32 to 1024 processors
  - Designed to scale to 16K processors
- Designed to be independent of specific processor node
- "Current" processor node
  - 40 MHz SPARC
  - 32 MB memory per node
  - 4 FP vector chips per node

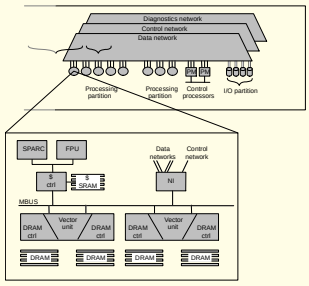
CM5, contd.

- **Vector Unit**
  - Four FP processors
  - Direct connect to main memory
  - Each has a 1 word data path
  - FP unit can do scalar or vector
  - 128 MFLOPS peak: 50 MFLOPS Linpack



Interconnection Networks

- **Input and output FIFO for each network**
- **Save/restore network buffers on context switch**



Interconnection Networks

- **Two networks: Data and Control**
- **Network Interface**
  - Memory-mapped functions
  - Store data => data
  - Part of address => control
  - Some addresses map to privileged functions

Message Management

- **Typical function implementation**
  - Each function has two FIFOs (in and out)
  - Two outgoing control registers:
    send and *send_first*
    send_first initiates message
    send sends any additional data
  - read *send_ok* to check successful send;else retry
  - read *send_space* to check space prior to send
  - incoming register *receive_ok* can be polled
  - read *receive_length_left* for message length
  - read *receive* for input data in FIFO order

Control Network

- **In general, every processor participates**
  - A mode bit allows a processor to *abstain*
- **Broadcasting**
  - 3 functional units: broadcast, supervisor broadcast, interrupt
    only 1 broadcast at a time
    broadcast message 1 to 15 32-bit words
- **Combining**
  - Operation types:
    reduction
    parallel prefix
    parallel suffix
    router-done (big-OR)
    Combine message is a 32 to 128 bit integer
    Operator types: OR, XOR, max, signed add, unsigned add
  - Operation and operator are encoded in *send_first* address

Control Network, contd

- **Global Operations**
  - Big OR of 1 bit from each processor
  - three independent units; one synchronous, 2 asynchronous
  - Synchronous useful for barrier synch
  - Asynchronous useful for passing error conditions independent of other Control unit functions
- **Individual instructions are not synchronized**
  - Each processor fetches instructions
  - Barrier synch via control is used between instruction blocks

    => support for a loose form of data parallelism

Data Network

- **Architecture**
  - Fat-tree
  - Packet switched
  - Bandwidth to neighbor: 20 MB/sec
  - Latency in large network:
  - 3 to 7 microseconds
  - Can support message passing or global address space
- **Network interface**
  - One data network functional unit
  - *send_first* gets destn address + tag
  - 1 to 5 32-bit chunks of data
  - tag can cause interrupt at receiver
- **Addresses may be physical or relative**
  - physical addresses are privileged
  - relative address is bounds-checked and translated

Data Network, contd.

- **Typical message interchange:**
  - alternate between pushing onto send-FIFO and receiving on receive-FIFO
  - once all messages are sent, assert this with combine function on control network
  - alternate between receiving more messages and testing control network
  - when *router_done* goes active, message pass phase is complete

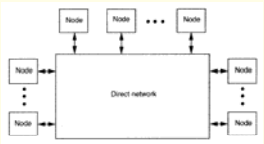
Lecture Summary

- **Introduction**
- **Software Scaling**
- **Hardware Scaling**
- **Case studies**
  - MIT J Machine
  - Cray T3D
  - Cray T3E*
  - CM-5**
- **Readings (to be discussed on Friday, 4/24)**
  - *) Steven L. Scott, Synchronization and Communication in the T3E Multiprocessor, Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pages 26-36, October 1996.
  - **) W. Daniel Hillis, Lewis W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer," CACM, pp. 30-40, Nov. 1993.

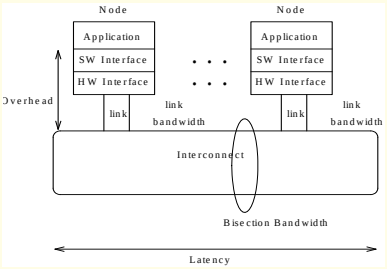
Interconnects & Routing (for reference)

Direct Networks

- **Point-to-point connections between nodes**
- **Scale well because each node adds**
  - memory capacity
  - memory bandwidth
  - some system bandwidth
- **Messages**
  - passed from source to destn by being relayed by intermediate nodes
  - may be explicitly programmer-directed
    or, they may be implicit (e.g. cache lines)
  - often broken into packets

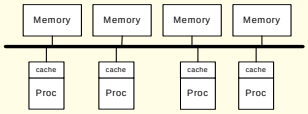


Interconnection Networks: Topology



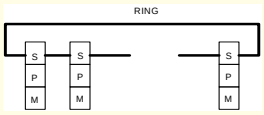
Buses

- Indirect or Direct interconnect
  - (Indirect version shown in Fig.)
- Performance/Cost:
  - Switch cost: N
  - Wire cost: const
  - Avg. latency: const
  - Bisection B/W: const
  - Not neighbor optimized
  - May be local optimized
- Capable of broadcast (good for MP coherence)
- Bandwidth not scalable ⇒ major problem
  - Hierarchical buses?
    Bisection B/W remains constant
    Becomes neighbor optimized



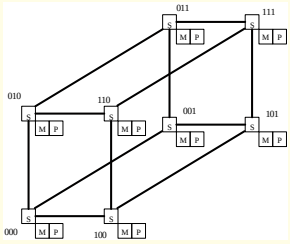
Rings

- Direct interconnect
- Performance/Cost:
  - Switch cost: N
  - Wire cost: N
  - Avg. latency: N / 2
  - Bisection B/W: const
  - neighbor optimized, if bi-directional
  - probably local optimized
- Not easily scalable
  - Hierarchical rings?
    Bisection B/W remains constant
    Becomes neighbor optimized



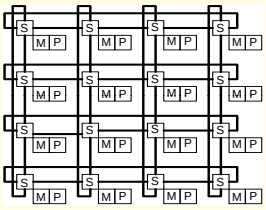
Hypercubes

- n-dimensional unit cube
- Direct interconnect
- Performance/Cost:
  - Switch cost: N $\log_2$ N
  - Wire cost: (N $\log_2$ N) /2
  - Avg. latency: ($\log_2$ N) /2
  - Bisection B/W: N /2
  - neighbor optimized
  - probably local optimized
- latency and bandwidth scale well, BUT
  - individual switch complexity grows
    ⇒ max size is often "built in" to switch design



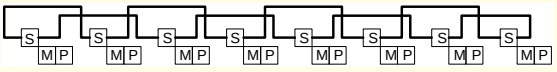
2D Torus

- Direct interconnect
- Performance/Cost:
  - Switch cost: N
  - Wire cost: 2 N
  - Avg. latency: N $^{1/2}$
  - Bisection B/W: 2 N $^{1/2}$
  - neighbor optimized
  - probably local optimized



2D Torus, contd.

- Cost scales well
- Latency and bandwidth do not scale as well as hypercube, BUT
  - difference is relatively small for practical-sized systems
- In physical design, can "weave" nodes to make inter-node latencies const.
- 2D Mesh similar, but without wraparound

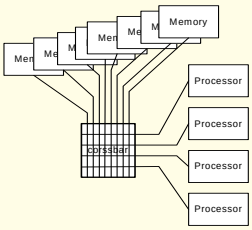


3D Torus

- General properties similar to 2D Torus
- Performance/Cost:
  - Switch cost: N
  - Wire cost: 3 N
  - Avg. latency: 3(N$^{1/3}$ / 2)
  - Bisection B/W: 2 N$^{2/3}$
  - neighbor optimized
  - probably local optimized
- 3D Mesh similar, but without wraparound
- Seem(ed) to be an interconnect of choice:
  - Cray, Intel, Tera, DASH, etc.
  - This may be changing…

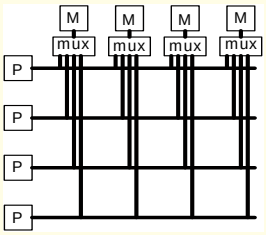
Crossbars

- Indirect interconnect
- Performance/Cost:
  - Switch cost: N^2
  - Wire cost: 2 N
  - Avg. latency: const
  - Bisection B/W: N
  - Not neighbor optimized
  - Not local optimized



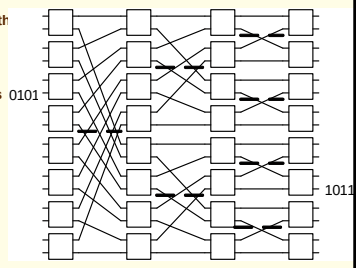
Crossbars, contd.

- Capable of broadcast
- No network conflicts
- Cost does not scale well
- Often implemented with muxes



Multistage Interconnection Networks (MINs)

- AKA: Banyan, Baseline, Omega networks, etc.
- Indirect interconnect
- Crossbars interconnected with shuffles
- Can be viewed as overlapped MUX trees
- Destination address specifies the path
- The shuffle interconnect routes addresses in a binary fashion
- This can most easily be seen with MINs in the form at right

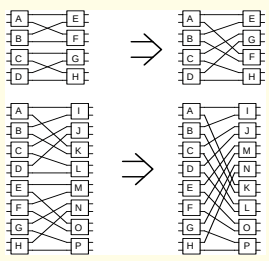


MINs, contd.

- f switch outputs ⇒ decode $\log_2 f$ bits in switch
- Performance/Cost:
  - Switch cost: $(N \log_f N) / f$
  - Wire cost: $N \log_f N$
  - Avg. latency: $\log_f N$
  - Bisection B/W: N
  - Not neighbor optimized
    - Can be a problem (in combination with latency growth)
  - Not local optimized
- Capable of broadcast
- Commonly used in large UMA systems
- Also used in large Vector MPs

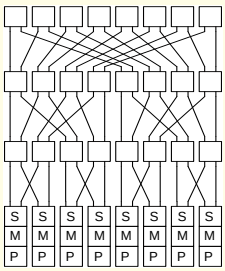
Multistage Nets, Equivalence

- By rearranging switches, multistage nets can be shown to be equivalent



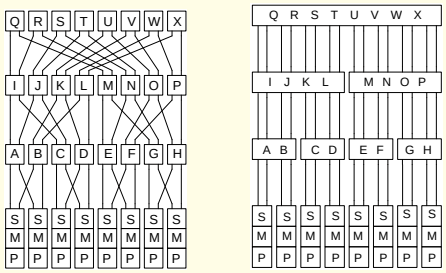
Fat Trees

- Direct Interconnect
- Tree-like, with constant bandwidth at all levels
- Closely related to MINs
- Indirect interconnect
- Performance/Cost:
  - Switch cost: $N \log_f N$
  - Wire cost: $f N \log_f N$
  - Avg. latency: approx $2 \log_f N$
  - Bisection B/W: f N
  - neighbor optimized
  - may be local optimized
- Capable of broadcast



Fat Trees, Contd.

- The MIN-derived Fat Tree, is, in fact, a Fat Tree:
- However, the switching "nodes" in effect do not have full crossbar connectivity



Summary

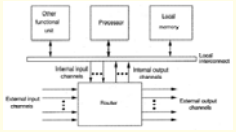
Computer	Year	Top.	link BW MB/sec	Bisec. BW MB/sec
TMC CM-2	1987	12-cube	1	1024
nCube/ten	1987	10-cube	1.2	640
Intel iPSC	1988	7-cube	2	345
Maspar MP-1216	1989	2D grid +MIN	3	1300
Intel Delta	1991	2D grid	40	640
TMC CM-5	1991	fat tree	20	10,240
Meiko CS-2	1992	fat tree	50	50,000
Intel Paragon	1992	2D grid	175	6400
Kend. Sq KSR-1	1992	Hierarch rings	32	64
IBM SP-2	1993	fat tree	40	20,480
Cray Rsch T3D	1993	3D torus	300	19,200
Cray Rsch T3E	1996	3D torus	600	153,600
IBM Blue Gene L	2003	3D torus	175	700,000

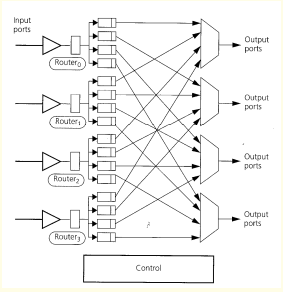
Routing

- Routing Methods
- Deadlock
- Deterministic Routing
- Adaptive Routing
- Virtual Channels

Routing

- Can be done by processors
  - requires an interrupt
  - consumes computing resources
- Or by special routers
  - This is commonly done in MPPs
- Typical node
  - Input, output, internal channel
  - Router may x-bar inputs and outputs, but not necessarily





Switching Techniques

- Circuit Switching
  - header packet establishes circuit (path)
  - circuit is reserved and packet is transmitted
  - circuit is torn down as the tail of the packet is transmitted

  => fast, once established,

  But, holds resources and may block other transmissions
- Packet Switching
  - path is established for each individual packet
  - there are a number of packet switching methods (next)

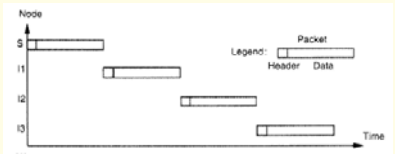
Store-and-Forward

- Packet is passed one "hop" at a time
- Packet is fully buffered in each node before being passed to the next

  => complete error checking can be done for each hop

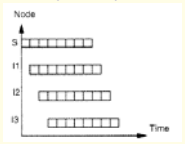
  => slow network latency
- Latency = L/B*D (L= packet length, B=channel bandwidth, D=no. hops)

  => strong function of distance



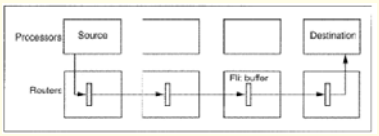
Virtual Cut-Through

- **Packet is broken into "flow control digits" (flits)**
  - Packet is passed one "hop" at a time
  - Flits move through network in pipelined fashion
  - If there is blockage, then packet is fully absorbed at blocking node
    => lots of buffering at each node
    => network communication resources are not tied up by blocking packet
- **Latency = D + L/B**
  - If L/B >>D, then latency is independent of distance



Wormhole routing

- **Like virtual cut-through, *except***
  - a packet is blocked "in place" at the various nodes its path
    => little buffering at each node
    => network communication resources *are* tied up by blocking packet
- **Wormhole routing is the current method of choice**



Routing Techniques

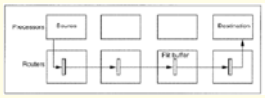
- **Source routing**
  - source determines the path
  - packet carries path info with it
- **Distributed routing**
  - each router contains routing info and decides next segment of route
  - this is the method of choice
- **Deterministic routing**
  - route is determined solely by the source and destn.
  - no flexibility for network load
- **Adaptive routing**
  - path determined by dynamic conditions
  - for example, network congestion

Flow Control

- **Techniques for dealing with resource contention in the network**
- **Relationship between routing and flow control**
  - Routing: output channel selection policy
  - Flow control: input channel selection policy
- **When an input is not selected:**
  - Drop
  - Block
  - Buffer
  - Re-route

Wormhole Routing, contd.

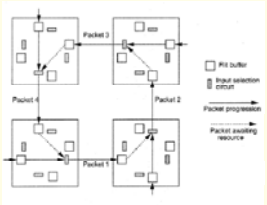
- **Channel is reserved for a packet and held until all flits pass through**
  - On blockage, trailing flits are buffered in intermediate nodes
  - Flit buffers may be small
  - Large buffers approach virtual cut-through
- **Wormhole routing allows a physical channel to be shared**
  => *virtual* channels
  - each virtual channel has a set of buffers



Deadlock

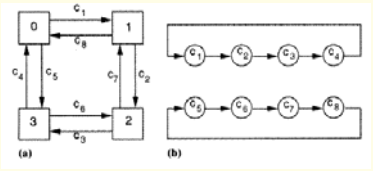
- **Holding/demanding of multiple resources by multiple tasks can lead to deadlock**
- **Wormhole routing holds channels, while requesting channels**
  => deadlock is a hazard
- **Solution: Pre-emption**
  - drop or re-route pre-empted packet (e.g. hot potato routing)
- **Solution: Avoidance**
  - Via routing algorithm
  - order resources (channels) and have packets use them in strict order

  => channel dependence graphs



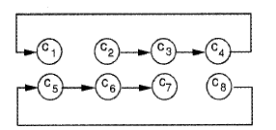
Channel Dependence Graphs

- **Graph for developing deadlock-free algorithms**
  - vertices: all unidirectional channels
  - edges: all pairs that may be connected by routing algorithm
- **A routing algorithm is deadlock free**
  - iff there is no cycle in the channel dependence graph
- **Example (Deadlock)**



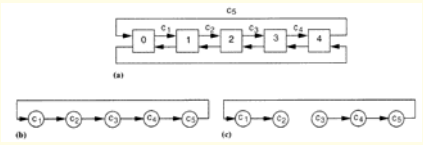
Dimension-Ordered Routing

- **Principle:**
  - Route one dimension at a time
  - Order the dimensions and all routes follow the same order
- **Example: e-cube routing in hypercubes**
  - XOR source and destn addresses
  - Go from left to right, routing along dimensions wherever there is a 1
- **Example: 2D mesh**
  - Route on X dimension first, then turn
  - Route on Y dimension next
- **Consider dependence graph for previous example**
  - no connections between: c5,c6; c4,c1; c7,c8; c2,c3
- **More flexible solutions are also possible:**



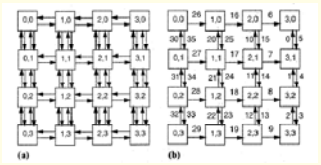
General k-ary n-cubes

- **If k>4, it is impossible to construct minimal deterministic deadlock free algorithm**
- **Example: 5-ary 1-cube (ring)**
  - A nonminimal deterministic route may be used (or virtual channels may be used -- later)



Adaptive Routing

- **Can adapt to network congestion**
- **Deadlock avoidance becomes more difficult**
  - => additional channels
- **Minimal adaptive routing**
  - partition channels into subnetworks
  - select subnetwork based on destn
  - routing within subnetwork can be adaptive
- **Generalizes, but added channels grow rapidly with n**

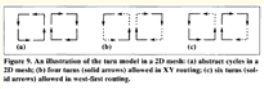


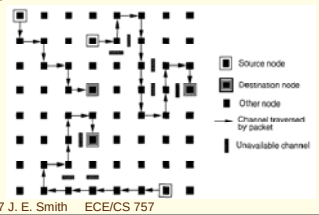
Nonminimal Adaptive Routing

- **In general: use r pairs of channels for each pair of connected nodes**
- **Static dimension reversal:**
  - partition into r identical subnetworks
  - class-i subnet has i-th pair of channels
  - packet header has class no., initially 0
  - if c < r-1, route at will on the class-c subnet
  - if route goes anti-dimension order, c++
  - once at class r-1, use strict dimension order
  - r determines the "degree of adaptivity"
- **Dynamic dimension reversal:**
  - partition into dynamic and deterministic subnets (to achieve some kind of balance)
  - Routing like static, but unlimited anti-dimension routes (however, c is incremented each time)
  - A packet blocked by messages all with lower values of c then switch to a deterministic subnet

Turn Model

- **Fundamental Concept:**
  - In dependence graphs, eliminate the smallest number of turns so that cycles are prevented
- **Example: 2-D mesh (also see earlier example)**
- **Routing alg: "west-first"**





Virtual Channels

- **Some of above methods add channels**
  => use virtual channels
- **A virtual channel has buffering and control logic**
- **Physical channels are multiplexed**
- **Bandwidth is shared among virtual channels**
  - Hardware should allocate bandwidth to active virtual channels
- **Advantages:**
  - physical interconnect can support richer set of logical interconnects
  - Allows easy prioritization of messages
- **Disadvantages:**
  - more hardware complexity in routing/scheduling
  - more latency due to routing/scheduling logic

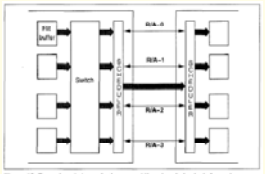


Figure 12. Four virtual channels share a unidirectional physical channel.