ECE/CS 757: Advanced Computer Architecture II Massively Parallel Processors

Instructor:Mikko H Lipasti

Spring 2017

University of Wisconsin-Madison

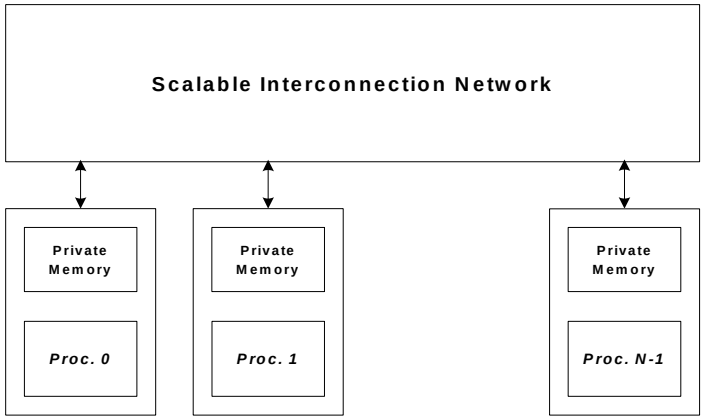
Lecture notes based on slides created by John Shen, Mark Hill, David Wood, Guri Sohi, Jim Smith, Natalie Enright Jerger, Michel Dubois, Murali Annavaram, Per Stenström and probably others

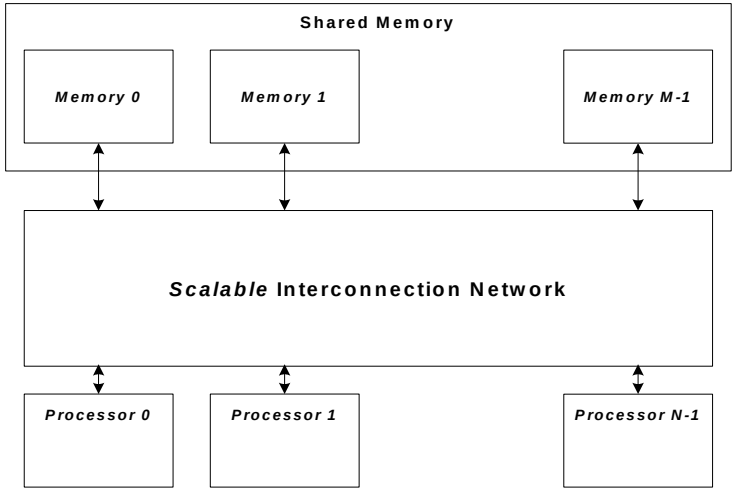
Lecture Outline

- Introduction
- Software Scaling
- Hardware Scaling
- Case studies
  - Cray T3D & T3E

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

    $\Rightarrow$ 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 processor chip
    - 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

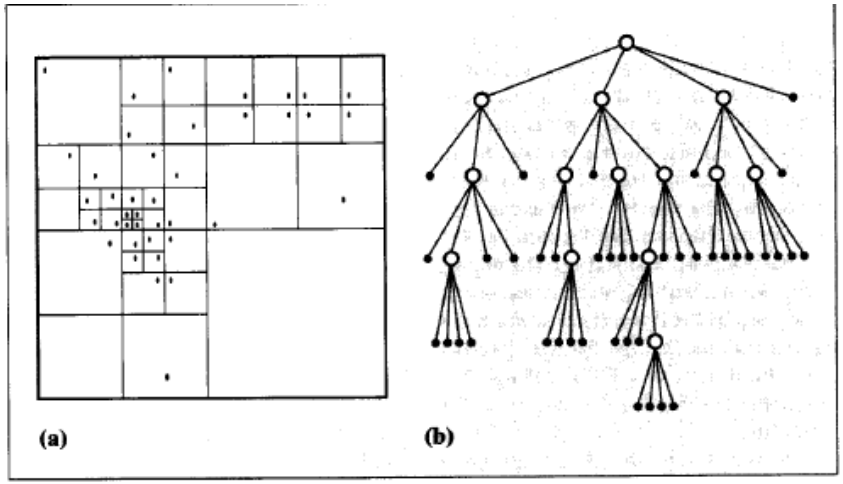
  Multiply matrices, parallel, complexity N^3

Problem Constrained Scaling

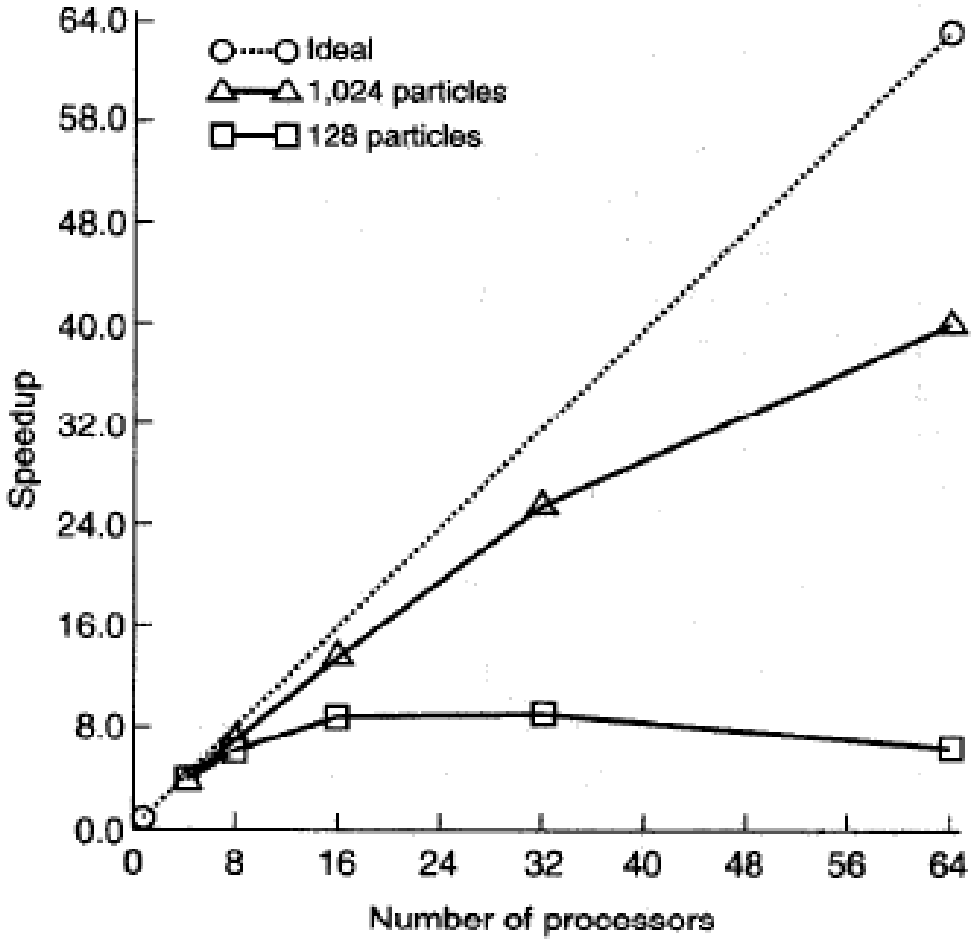
- User wants to solve same problem, only faster
  - E.g., Video compression
- Amdahl's law is a limitation
- In many cases, problem sizes grow
  - Reevaluating Amdahl's Law, John L. Gustafson, Communications of the ACM 31(5), 1988. pp. 532-533.

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

  $S(1) = 1$

  $S(10) \sim 10$

  $S(100) \sim 100$

  $S(1000) \sim 1000$

- BUT, 1000 times increase in problem size $\Rightarrow$

  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
    - Render next frame in real time at 60fps
- 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 $N^{1/3}$

  $\Rightarrow$ 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 $\Rightarrow$ 100 elements per 1 processor
  - Scale up to 1000 processors $\Rightarrow$ 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 at different rates
  - 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 $k^{3/4}$

    kn reduces error by sqrt(k)

    Error is ~dt^2 so to get equal error scaling of sqrt(k) must decrease dt by by factor of k^0.25 (4th root)

    Error is ~(fa)^2 so to get equal error scaling of sqrt(k) must decrease (fa) by factor of k^0.25

    These in combination cause k^3/4 increase

Hardware Scaling

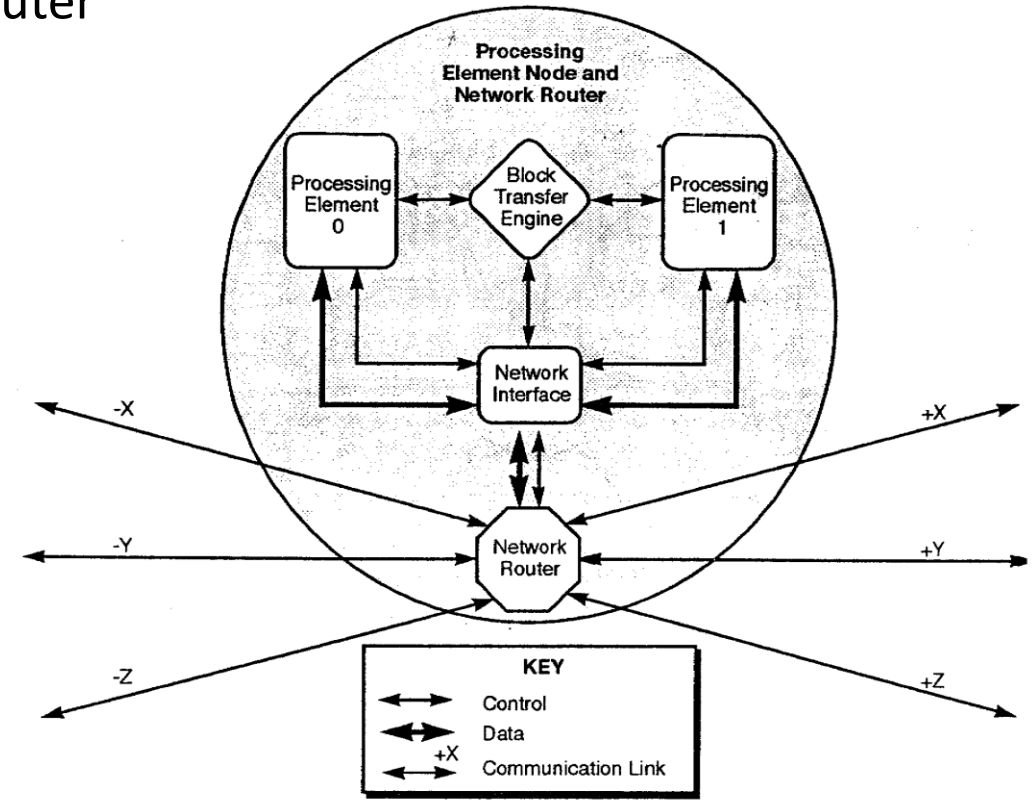
- Bandwidth Scaling
  - should increase proportional to # procs.
  - crossbars
  - multi-stage nets
- Latency Scaling
  - ideally remain constant
  - in practice, logn scaling can be achieved
  - local memories help (local latency may be more important than global)
  - latency may be dominated by overheads, anyway
- Cost Scaling
  - low overhead cost (most cost incremental)
  - in contrast to many large SMPs
- Physical Scaling
  - loose vs. dense packing, 2D vs. 3D
  - chip level integration vs. commodity parts

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

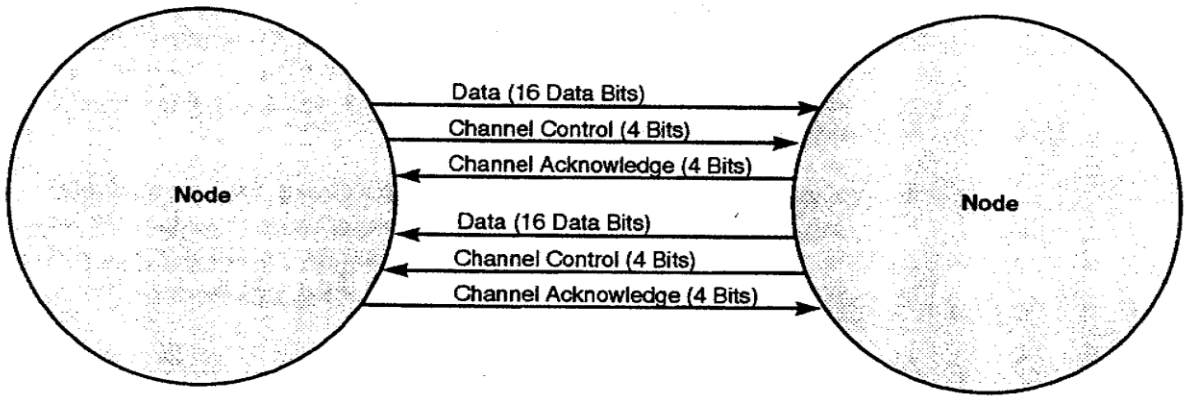
Processing Element Nodes

- 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

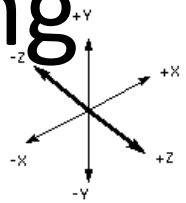


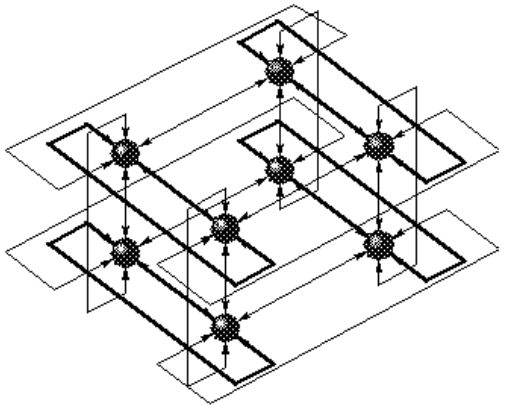
Communication Links

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



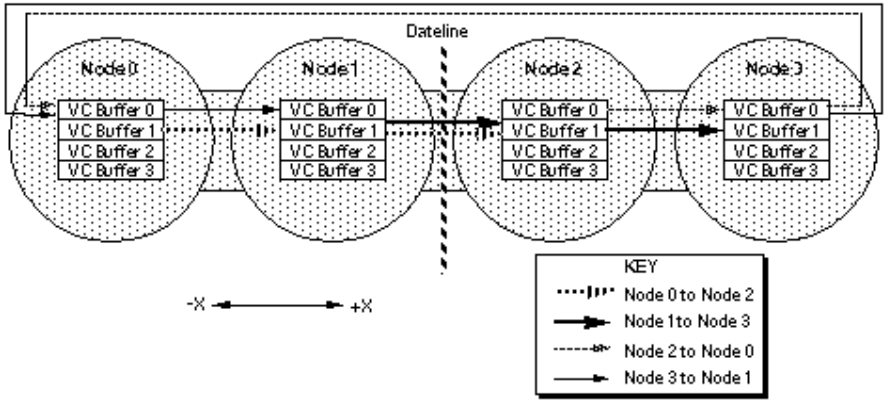
Routing





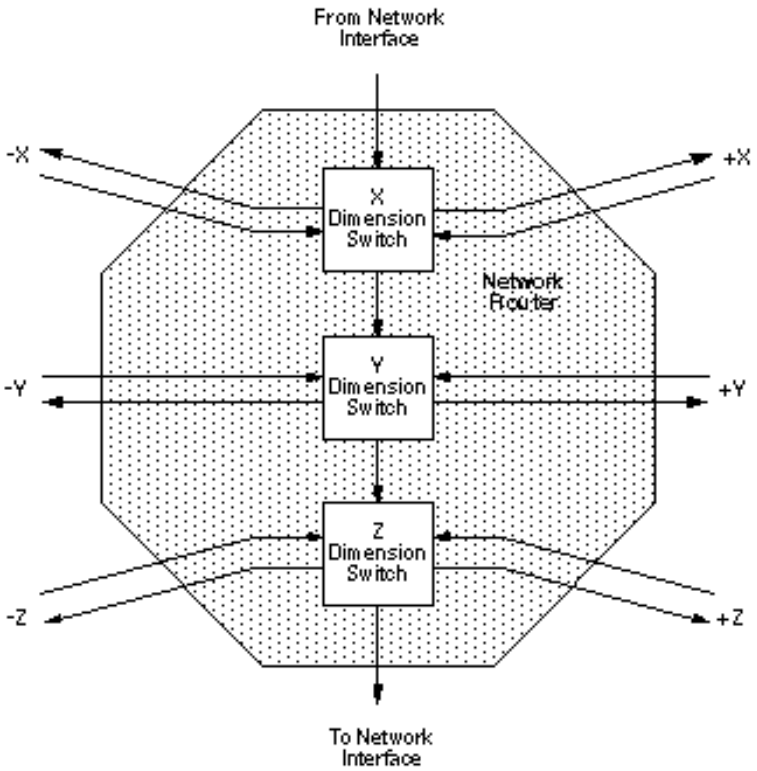
- 3D Torus
- Dimension order routing
  - may go in either + or - direction along a dimension
- Virtual channels
  - Four virtual channel buffers per physical channel

    $\Rightarrow$ 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



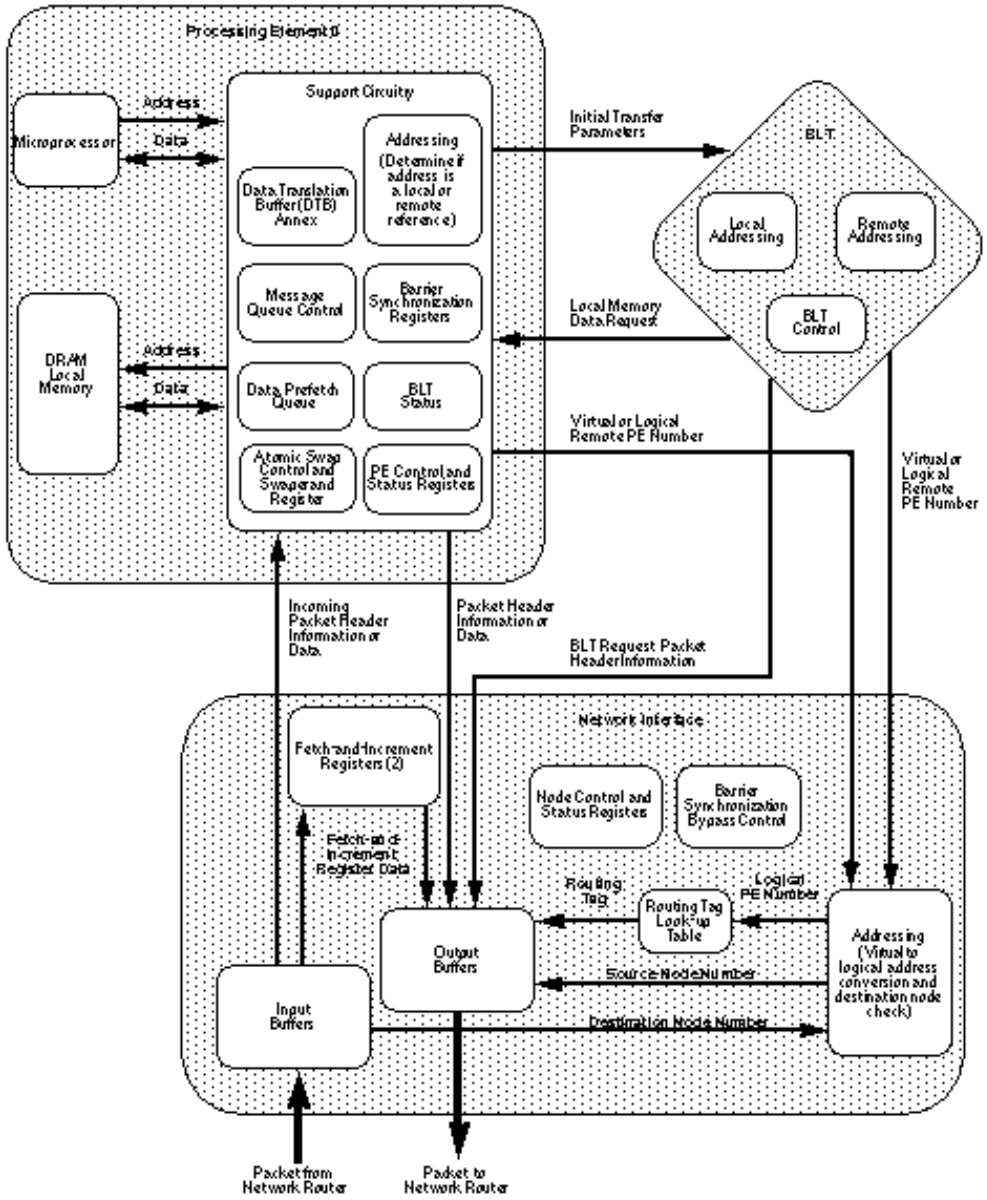
Network Routers

- Separate X,Y,Z dimensions



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)

$\Rightarrow$ 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

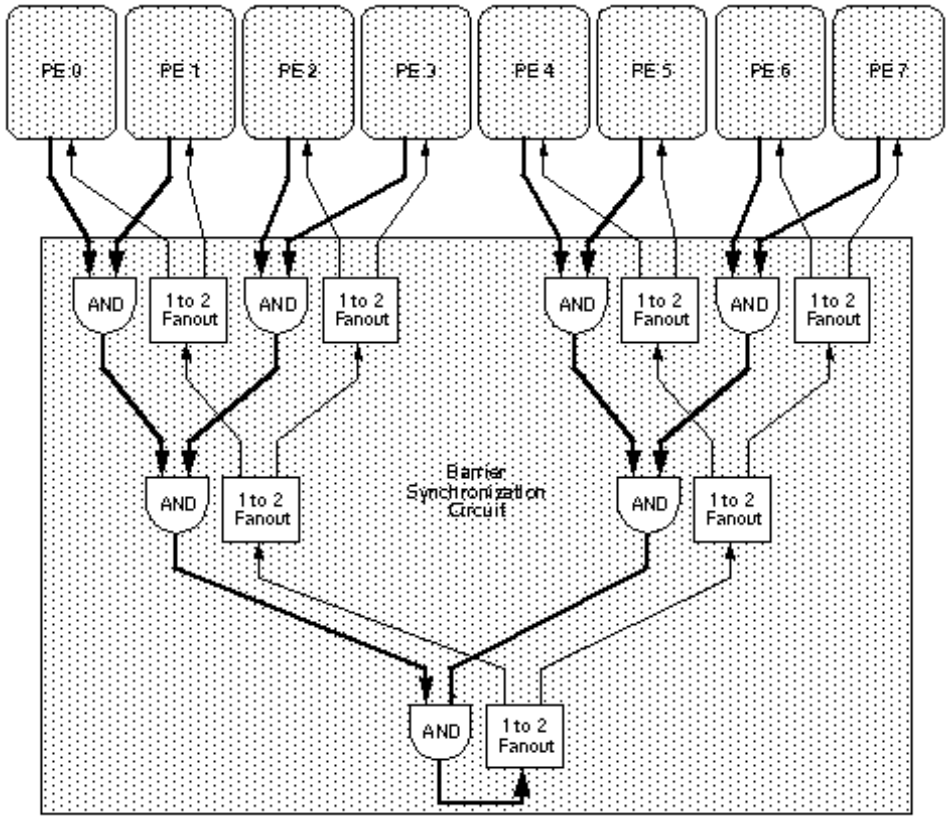
- Architected Prefetch Queue
  - 1 word wide by 16 deep
- Prefetch instruction:
  - Alpha prefetch hint => T3D prefetch
- Performance
  - Allows multiple outstanding read requests
  - (normal 21064 reads are blocking)

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

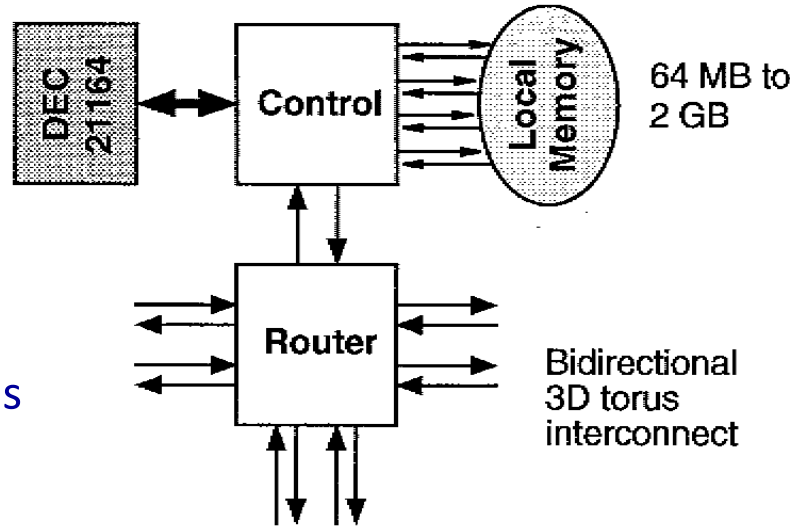
- T3D Post Mortem
- T3E Overview
- Global Communication
- Synchronization
- Message passing
- Kernel performance

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

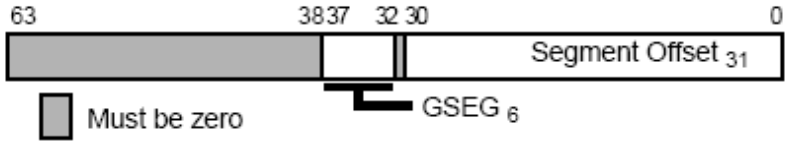
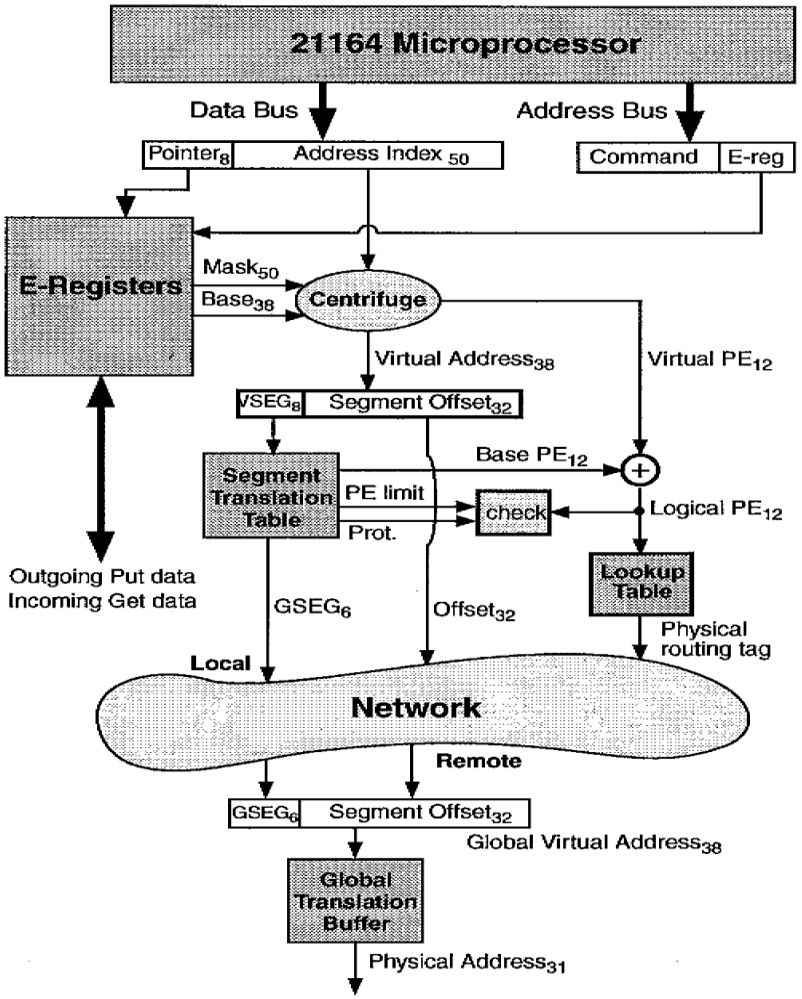


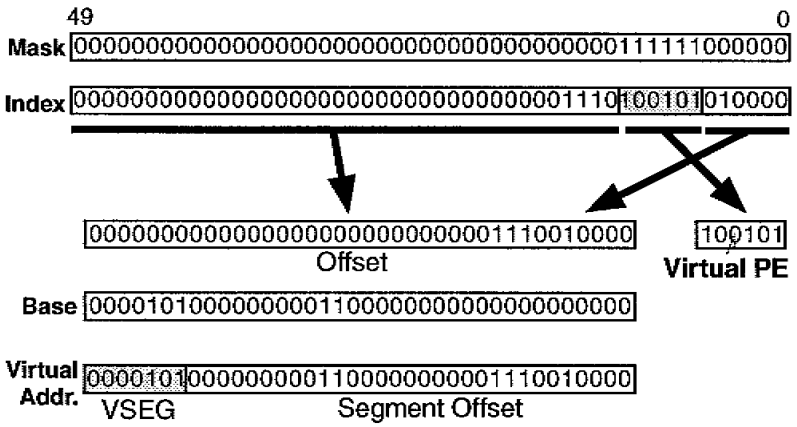
Figure 2. Global virtual address (GVA)

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

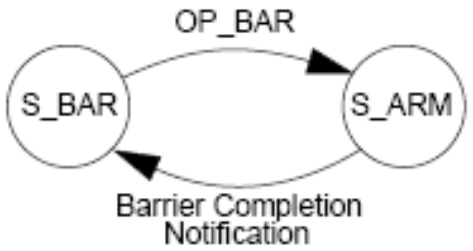
- Atomic ops between E-regs and memory
  - Fetch & Inc
  - Fetch & Add
  - Compare & Swap
  - Masked Swap
- Barrier/Eureka Synchronization
  - 32 BSUs per processor
  - accessible as memory-mapped registers
    - protected via address translation
  - BSUs have states and operations
  - State transition 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

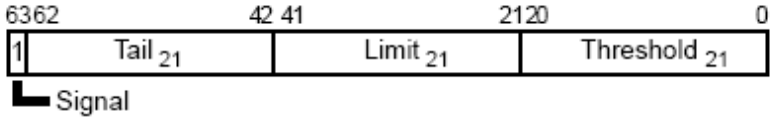


Figure 5. Message Queue Control Word

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

Recent Trends

- High-end microprocessors now include vector FP & high-BW DRAM
  - Intel, AMD: SSEx/AVX
  - DDRx evolution, on-chip MCs
  - High-performance coherent caches: HT, QPI
- Higher-throughput FP also available through PCIe GPGPUs
  - Nvidia's CUDA, AMD OpenCL
- Commodity NICs also provide high BW, low latency
  - 10Gbit/25Gbit/40Gbit ethernet, RDMA support
  - High-radix routers enable low-diameter datacenter topologies
- Clusters built out of:
  - 2 socket SMP nodes, 16c/32t or more + GPGPU

Lecture Summary

- Introduction
- Software Scaling
- Hardware Scaling
- Case studies
  - Cray T3D/Cray T3E