Announcements

- Project proposals due today
- Homework 3 due back on Wednesday, October 24th
  - Parameters for performance measurements posted on mailing list
  - I may not have Internet access from October 21st – October 25th
- No lecture next Wednesday (October 24th)
  - Turn in Homework 3 by e-mail/put it in my mailbox
  - Homework 4 will be available for download from the web site

Outline

- Last lecture
  - Architecture of symmetric multiprocessors (cont’d)
    - synchronization
    - case study: Sun Enterprise
  - Architecture of distributed memory multiprocessors

- This lecture
  - Architecture of distributed memory multiprocessors (cont’d)
    - Supporting various programming models
    - Hardware design choices
    - Case studies: Cray T3E, IBM SP
  - Large-scale shared memory machines
    - Extending bus-based coherence
    - Directory-based coherence schemes

[Culler/Singh/Gupta: Chapters 7-8, web sources]
Supporting a Shared Address Space

- Issues
  - *input buffer overflow*: many nodes request from same address
  - *fetch deadlock*: a node must be able to sink requests
  - *delivery order*: important for implementing memory consistency models
    (more about this later)

Message Passing: Asynchronous Protocols

- Problems with the optimistic protocol
  - *store-and-forward delay* (if no posted receive buffer)
  - *input buffer overflow* (function of program behavior)

For small messages, a *credit-based* scheme can reduce handshake costs
Active Messages

- A message-passing abstraction closer to the level of the underlying network transaction
  ```
  send( node, handler, buf, len )
  extract( )
  ```
  - message send associated with a handler at the other end
    - handler integrates message into ongoing computation
    - allocates a buffer to store message if necessary
      - very often, this is not required (e.g., for looking up an address)
  - no explicit receive
    - reduces cost of tag matching and buffering
  - request-response transactions constitute a restricted RPC
    - response handlers cannot send messages to prevent deadlock
  - interface permits very efficient implementations
    - Higher-level abstractions (MPI) typically built on top of active messages

- More details in Lecture 11

---

HW Design Choices (1): Physical DMA

- HW does no interpretation on the information within a transaction
  - representative of early message-passing machines
    - nCUBE10, nCUBE/2, Intel iPSC, iPSC/2, iPSC860, Delta, Ametek, SP-1
  - physical DMA to/from network
    - use of physical addresses
    - sending/receiving requires OS intervention

---

HW Design Choices (2): User-level Access

- HW distinguishes between system and user messages
  - user messages both injected and received without OS intervention
    - more recent machines: CM-5, Cray T3D/T3E, VIA

---

HW Design Choices (3): Dedicated Message Processing

- HW dedicates resources to interpret information in transactions
  - protocol processing can be off-loaded to communication processor
    - example machines: Meiko CS-2, Intel Paragon
  - differentiated by whether CP is a symmetric processor, or an embedded processor (with its own path to the network interface)
HW Design Choices (4):
Shared Physical Address Space

- HW provides support for loads, stores, atomic operations
  - dancehall: NYU Ultracomputer, BBN Butterfly, IBM RP-3
  - distributed memory: Cray T3D, Cray T3E, Origin 2000

Clusters and Networks of Workstations

- Building scalable machines using commodity components
  - Systems: processors, memory, disk (e.g., a PC)
  - High-performance networks (e.g., Myrinet, VIA)
    - connects to the system on the peripheral bus
  - Cost-effective, but needs to overcome several disadvantages
    - Loose coupling of network to node
    - Independent task scheduling
    - ...

Implications for Parallel Software

- Network transaction performance: \( T(n) = T_0 + \frac{n}{B} \)

<table>
<thead>
<tr>
<th>Machine</th>
<th>Year</th>
<th>MFLOPs</th>
<th>( T_s ) (µs)</th>
<th>( T_c ) (cycles)</th>
<th>( T_f ) (FLOPs)</th>
<th>( B ) (MB/s)</th>
<th>( m/2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPSC/2</td>
<td>1988</td>
<td>1</td>
<td>700</td>
<td>5600</td>
<td>700</td>
<td>2.7</td>
<td>1400</td>
</tr>
<tr>
<td>nCUBE/2</td>
<td>1990</td>
<td>2</td>
<td>160</td>
<td>3000</td>
<td>300</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>iPSC/860</td>
<td>1991</td>
<td>40</td>
<td>160</td>
<td>6400</td>
<td>6400</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>CM-5</td>
<td>1992</td>
<td>20</td>
<td>95</td>
<td>3135</td>
<td>1900</td>
<td>8</td>
<td>760</td>
</tr>
<tr>
<td>SP-1</td>
<td>1993</td>
<td>100</td>
<td>50</td>
<td>2500</td>
<td>5000</td>
<td>25</td>
<td>1250</td>
</tr>
<tr>
<td>Meiko CS-2</td>
<td>1993</td>
<td>24</td>
<td>83</td>
<td>7470</td>
<td>1992</td>
<td>43</td>
<td>3560</td>
</tr>
<tr>
<td>Paragon</td>
<td>1994</td>
<td>50</td>
<td>30</td>
<td>1500</td>
<td>1500</td>
<td>175</td>
<td>7240</td>
</tr>
</tbody>
</table>

- overlap possible between components of transaction
  - \( O_s \) and \( O_c \) (overhead at sender and receiver), and \( L \) (network latency)

Case Study (1): Cray T3E (Tightly Coupled MP)

DEC Alpha 21164
300-600 MHz
8KB L1 instr and data
96 KB L2 unified
2 outstanding memory refs

Control
Shell Logic

Router, 3D interconnect
600 MB/s per link in each direction

Local Memory
64MB to 2GB

2048 processors, 3D Torus
Cray T3E: The “Glue” Logic

- Central mechanism of the “glue” logic
  - 512 user + 128 system registers (64-bit)
  - memory-mapped into non-cacheable I/O space
    - bit 39 of physical address distinguishes cacheable/noncacheable space
  - 2 types of operations
    - implicitly synchronized using full/empty flags
    - global E-register operations (gets and puts) use I/O space stores
      - address encodes command (put/get), and src/dest E-register
      - data encodes pointer to block of E-registers and address index
        » flexible data distribution
        » split-phase operations: 4 put/get operations every 2 (75 MHz) cycles
      - strided puts/gets
  - Benefits
    - pipelined remote memory operations
    - high memory-to-memory bandwidth
      - 60 MB/s versus 340 MB/s for random gather

Cray T3E: E-Registers

- Central mechanism of the “glue” logic
  - 512 user + 128 system registers (64-bit)
  - memory-mapped into non-cacheable I/O space
    - bit 39 of physical address distinguishes cacheable/noncacheable space
  - 2 types of operations
    - implicitly synchronized using full/empty flags
    - global E-register operations (gets and puts) use I/O space stores
      - address encodes command (put/get), and src/dest E-register
      - data encodes pointer to block of E-registers and address index
        » flexible data distribution
        » split-phase operations: 4 put/get operations every 2 (75 MHz) cycles
      - strided puts/gets

Cray T3E: Messaging

- Arbitrary number of user-level message queues
  - 64-byte messages
  - queues can be any size up to 128 MB
- Message queue control word (MQCW)

<table>
<thead>
<tr>
<th>S</th>
<th>Tail</th>
<th>Limit</th>
<th>Threshold</th>
</tr>
</thead>
</table>
- store MQCW at desired location of the queue
- user responsible for buffer management
- Threshold determines when processor is interrupted (S is set)
- Sending a message
  - assemble message in block of 8 E-registers
  - issue a SEND command (address must point to valid MQCW)
  - flags on source E-registers indicate success/failure
- Performance
  - 5µs latency, ~350MB/s bandwidth

Cray T3E: Synchronization Operations

- Atomic operations against arbitrary memory locations
  - Build upon E-register support
    - specify AMO command on address bus
    - operands taken from aligned block of base/index E-registers (data bus)
    - result returned to E-register specified on address bus
  - Operations
    - Fetch&Inc (hardware merge support)
    - Fetch&Add
    - Compare&Swap
    - Masked_Swap
  - Performance
    - Sustained rate: 40ns per F&I, 222ns per op for the others
    - Latency: ~1.5µs
- Hardware barrier support
  - Single cycle check
Case Study (2): IBM SP

- Relatively loosely coupled
- Five types of nodes
  - “Thin” and “Wide” 332 MHz PowerPC 604e nodes
    - 2–4 CPUs per board
  - “Thin”, “Wide”, and “High” 375 MHz Power 3 nodes
    - 2–4 CPUs per board in the first two (difference in no. of PCI slots)
    - 16 per board in the last one
- Different network options
  - Ethernet, Token Ring, FDDI, high-performance SP-2 interconnect
  - SP-2 switch performance
    - Latency: 20 µs
    - Bandwidth: 140 – 350 MB/s

(Recap) Supporting a Physical Shared Address Space on Distributed Memory Multiprocessors

- Natural to fetch and cache entire line
- However, need to keep cached data coherent
  - protocols cannot do broadcast and match (snoop!)

Extending Bus-based Coherence Schemes

- Motivation
  - leverage SMP building blocks, packaging, optimized technology
  - preserve tight cluster interaction
- Issues
  - can these systems be put together (without changing anything inside)
  - what kind of overheads do these systems incur?
- Several examples
  - Encore Gigamax, Convex Exemplar, Sequent NUMA-Q
Hierarchy of Busses

- Snoopy cache protocol
  - on B1 with L2 acting as memory, on B2 with L2 acting as processor

- Scaling considerations
  - number of processors: limited by packaging
  - memory latency and bandwidth: single traversal to root
  - coherency protocol bandwidth: many B1, one B2
  - locality/placement: sharing locality avoids B2 broadcasts

Cluster-based Hierarchies (Encore Gigamax)

- Main memory distributed among clusters
  - L2 cache can be replaced by a tag-only router-coherence switch

- Scaling considerations
  - number of processors: limited by packaging
  - memory latency and bandwidth: multiple, fast local access
  - coherency protocol bandwidth: many B1, one B2
  - locality/placement: important

Cache Coherence in Encore Gigamax

- Router-coherence switch must know about
  - local memory words in remote caches and their state (clean/dirty)
  - remote memory words in local caches and their state

- Operation
  - write to B1 is passed to B2 if
    - reference to a remote memory word
    - reference to a local memory word, but present in some remote cache
  - read to B1 is passed to B2 if
    - reference to a remote memory word (and not in cluster cache)
    - reference to a local memory word, but dirty in some remote cache
  - write to B2 is passed to B1 if
    - reference to a local memory word
    - data belongs to remote memory, but the block is dirty in a local cache
  - ...
  - many race conditions possible: write-back going out as request coming in

Ring-based Cache Coherence (KSR)

- Buses can be replaced by rings
  - any media capable of a broadcast

- Scaling considerations
  - number of processors: many
  - memory latency and bandwidth: linear in P
  - coherency protocol bandwidth:
    - broadcast medium, but no global serialization
    - many concurrent transactions, out of phase (many, many states)
  - locality/placement: within B1, all same on R2
Lessons from Hierarchical Coherence Schemes

- Why does bus-based coherence work?
  - FSM sequences with effectively atomic transitions ensure consensus on status of memory block and therefore coherence.

- Why do extensions to bus-based schemes work?
  - Layers extend these FSM transitions, delaying additional accesses until a global decision can be enforced.
  - Scaling limitations, but the overall scheme works because of global agreement.

- General formulation: “directory-based” structure
  - Associate an explicit state with each memory block.
  - Query and update this state using atomic transitions.
    - Memory consistency is ensured by restricting what this state can be.

Directory-based Cache Coherence

- Snoopy schemes do not scale because they rely on broadcast.
  - Hierarchical snoopy schemes have root as a bottleneck.

- Directory-based schemes allow scaling.
  - Avoid broadcasts by keeping track of all PEs caching a memory block.
  - Coherence maintained using point-to-point messages.
  - Allow flexibility to use any scalable point-to-point network.

A Simple Directory-based Coherence Scheme

- Full bit-vector (Censier and Feautrier, 1978)

  - Read by PE\textsubscript{i}:
    - If dirty bit is OFF: 
      - Read from main memory; turn p[i] ON.
    - If dirty bit is ON:
      - Recall line from dirty PE (change state to shared); update memory; turn dirty bit OFF; turn p[i] ON; supply recalled data to PE\textsubscript{i}.

A Simple Directory-based Scheme (cont’d)

- Full bit-vector (Censier and Feautrier, 1978)

  - Write by PE\textsubscript{i}:
    - If dirty bit is OFF: 
      - Supply data to PE\textsubscript{i}; send invalidations to all PEs caching that block; turn dirty bit ON; turn p[i] ON.
    - If dirty bit is ON:
      - Recall line from dirty PE (change state to invalid); update memory; turn p[i] ON; supply recalled data to PE\textsubscript{i}. 

Directory-based Coherence: Key Issues

- What information is needed to achieve coherence?
  - a means to trap accesses from processors, and force these global memory system operations to happen correctly
  - a means to purge/revive other processors’ local copies
    - keep track of where the copies are
    - multicast invalidation requests
  - global state transitions for the block, similar consistency as with the bus-based approach

- Scaling considerations
  - number of processors: many
  - memory latency and bandwidth: scalable network
  - coherence protocol bandwidth: ?
  - function of how many concurrent invalidates (updates), number of copies
  - locality/placement: moderately important
    - local network transactions

Cache Invalidation Patterns

- Hypothesis
  - on a write to a shared location, with high probability, only a small number of caches need to be invalidated

- If the above were not true, directory schemes would offer little advantage over snooping schemes!

- Empirical study (see Culler/Singh/Gupta: Section 8.3.1)
  - SPLASH-2 benchmarks running on 64 processors
  - infinite capacity, fully associative caches
    - cache replacement might reduce number of sharers

Invalidation Patterns

- Different categories of sharing
  - code and read-only objects (e.g., A and B matrices in matrix multiply)
    - no problem since never written
  - migratory objects (e.g., global sum onto which PEs add their partial sums)
    - only a single invalidation generated per write, independent of P
  - mostly-read objects (e.g., a bound variable in branch-and-bound TSP)
    - invalidations are large but infrequent: so little impact on performance
  - frequently read/written objects (e.g., task queues)
    - invalidations usually remain small, though frequent
  - producer-consumer sharing (e.g., near-neighbor interactions in eqn. solver)
    - typically, few invalidations
  - synchronization objects
    - low-contention locks result in few invalidations
    - high-contention locks need special support
      - hardware combining (NYU Ultracomputer)
      - software trees, queuing locks
Performance Issues

- How long do each of the protocol operations take?
  - read a global directory, locate and purge copies, change state, move copies of data
  - typical solutions:
    - exploit concurrency across operations
    - different memory blocks, multiple invalidations
    - relaxed consistency models
    - reduce need for protocol operations
    - pipeline requests and acknowledgements
  - more details in Lecture 7

- How much memory is needed for all these states/protocols?
  - store block’s consistency state, locate copies, state for ongoing transactions
  - is memory proportional to physical memory, cache sizes, or something else?

Directory Organizations

- Memory-based schemes
  - e.g., Stanford DASH, FLASH, MIT Alewife, SGI Origin 2000
  - directory storage proportional to memory blocks
  - full-map vs. partial-map (limited pointers)
    - main issue here is dealing with overflow
  - dense vs. sparse
    - directory is itself a cache

- Cache-based schemes
  - e.g., SCI (Sequent NUMA-Q, Convex Exemplar)
  - use cache blocks to link together sharing chain
    - storage scales with total amount of cache
    - insertion, deletion, communication
      - single vs. double link
    - built out of SRAM (faster)
    - more messages than memory-based schemes

Analysis of Memory-based Schemes

- Full-bit vector
  - storage
    - one bit of directory memory per main-memory block per PE
    - memory requirements = \( P \cdot \frac{P \cdot M}{B} \)
    - where \( P \) is the number of processors, \( M \) is main memory per PE, and \( B \) is cache-block size
    - overhead not too large for medium-scale MPs
      - e.g., 256 PEs organized as 64 4-PE clusters, 64 byte cache blocks
      - overhead = 64 bits for each 64-byte block (12.5% overhead)

  - invalidation traffic
    - less than limited pointer schemes

  - one way to reduce overhead is to increase \( B \)
    - can result in false sharing and increased coherence traffic
    - ideally, would like different block sizes based on sharing behavior
      - will become possible with reconfigurable architectures

Limited Pointer Schemes

- Rationale: Few sharers most of the time, so keep few pointers
- Distinguished based on overflow scheme
  - DIR-i-B
    - beyond i-pointers, set invalidate-broadcast bit to ON
    - memory requirements = \( i \cdot \log \frac{P \cdot M}{B} \)
    - works well if sharing behavior is in one of two extremes: few sharers, or lots of sharers

  - DIR-i-NB
    - when sharers exceed i, invalidate one of the existing sharers
    - significant degradation expected for widely shared, mostly-read data

  - DIR-i-CV-r
    - when sharers exceed i, use bits allocated to i pointers as a coarse-resolution vector (each bit points to multiple PEs)
    - always results in less traffic than DIR-i-B

- Limitless directories (Alewife): Handle overflow using software traps
Sparse Directories

- **Rationale**: Since total number of cache blocks is much less than total number of memory blocks, most directory entries are idle most of the time
  - e.g., 256 KB cache, 16 MB memory per PE  =>  >98% idle

- Sparse directories reduce memory requirements by
  - using single directory entry for multiple memory blocks (as in a cache)
    - directory entry can be freed by invalidating cached copies of a block
    - main problem is the potential for excessive directory entry conflicts
    - solution: associative sparse directories (as in a cache)!

Summary

- This lecture
  - Scalable distributed memory multiprocessors
    - Support for different programming models
    - Hardware design issues
    - Case studies: Cray T3E, IBM SP
  - Scalable shared memory multiprocessors
    - Extensions to bus-based coherence schemes
    - Directory-based coherence

- Next lecture (November 2nd, 2001)
  - Scalable shared memory machines (cont’d)
    - Relaxed consistency models
    - Case study: SGI Origin 2000
  - Software shared memory models