Distributed Memory Multiprocessors

Fundamental Issues

- 3 Issues to characterize parallel machines
  1) Naming
  2) Synchronization
  3) Performance: Latency and Bandwidth (covered earlier)

Fundamental Issue #1: Naming

- Naming:
  - what data is shared
  - how it is addressed
  - what operations can access data
  - how processes refer to each other
- Choice of naming affects code produced by a compiler; via load where just remember address or keep track of processor number and local virtual address for msg. passing
- Choice of naming affects replication of data; via load in cache memory hierarchy or via SW replication and consistency

Fundamental Issue #2: Synchronization

- To cooperate, processes must coordinate
- Message passing is implicit coordination with transmission or arrival of data
- Shared address
  + additional operations to explicitly coordinate: e.g., write a flag, awaken a thread, interrupt a processor
Parallel Architecture Framework

- **Programming Model**
  - Multiprogramming: lots of jobs, no communication
  - Shared address space: communicate via memory
  - Message passing: send and receive messages

- **Communication Abstraction**
  - Shared address space: e.g., load, store, atomic swap
  - Message passing: e.g., send, receive library calls

**Communication Abstraction**

- Debate over this topic (ease of programming, scaling)
  - Many hardware designs 1:1 programming model

Scalable Machines

- **What are the design trade-offs for the spectrum of machines between?**
  - Specialize or commodity nodes?
  - Capability of node-to-network interface
  - Supporting programming models?

- **What does scalability mean?**
  - Avoids inherent design limits on resources
  - Bandwidth increases with P
  - Latency does not
  - Cost increases slowly with P

Bandwidth Scalability

- What fundamentally limits bandwidth?
  - Single set of wires
- Must have many independent wires
- Connect modules through switches
- Bus vs Network Switch?

Dancehall MP Organization

- Network bandwidth?
- Bandwidth demand?
  - Independent processes?
  - Communicating processes?
- Latency?

Generic Distributed Memory Org.

- Network bandwidth?
- Bandwidth demand?
  - Independent processes?
  - Communicating processes?
- Latency?

Key Property

- Large number of independent communication paths between nodes
  - Allow a large number of concurrent transactions using different wires
- Initiated independently
- No global arbitration
- Effect of a transaction only visible to the nodes involved
  - Effects propagated through additional transactions
Programming Models Realized by Protocols

- CAD
- Database
- Scientific modeling
- Parallel applications

- Multiprogramming
- Shared address space
- Message passing
- Data parallel

- Communication abstraction
- User/system boundary
- Operating systems support
- Hardware/software boundary

Network Transaction

- Key Design Issue:
  - How much interpretation of the message?
  - How much dedicated processing in the Comm. Assist?

Shared Address Space Abstraction

- Fundamentally a two-way request/response protocol
  - writes have an acknowledgment

- Issues
  - fixed or variable length (bulk) transfers
  - remote virtual or physical address, where is action performed?
  - deadlock avoidance and input buffer full

- coherent? consistent?

Key Properties of Shared Address Abstraction

- Source and destination data addresses are specified by the source of the request
  - a degree of logical coupling and trust
- no storage logically “outside the address space”
  - may employ temporary buffers for transport
- Operations are fundamentally request response
- Remote operation can be performed on remote memory
  - logically does not require intervention of the remote processor

Consistency

- write-atomicity violated without caching

Message passing

- Bulk transfers
- Complex synchronization semantics
  - more complex protocols
  - More complex action

- Synchronous
  - Send completes after matching recv and source data sent
  - Receive completes after data transfer complete from matching send

- Asynchronous
  - Send completes after send buffer may be reused
Synchronous Message Passing

- Constrained programming model.
- Deterministic! What happens when threads added?
- Destination contention very limited.
- User/System boundary?

Asynch. Message Passing: Optimistic

- More powerful programming model
- Wildcard receive => non-deterministic
- Storage required within msg layer?

Asynch. Msg Passing: Conservative

- Where is the buffering?
- Contention control? Receiver initiated protocol?
- Short message optimizations

Key Features of Msg Passing Abstraction

- Source knows send data address, dest. knows receive data address
  - after handshake they both know both
- Arbitrary storage “outside the local address spaces”
  - may post many sends before any receives
  - non-blocking asynchronous sends reduces the requirement to an arbitrary number of descriptors
  » fine print says these are limited too
- Fundamentally a 3-phase transaction
  - includes a request / response
  - can use optimistic 1-phase in limited “Safe” cases
    » credit scheme

Active Messages

- User-level analog of network transaction
  - transfer data packet and invoke handler to extract it from the network and integrate with on-going computation
- Request/Reply
- Event notification: interrupts, polling, events?
- May also perform memory-to-memory transfer

Common Challenges

- Input buffer overflow
  - N-1 queue over-commitment => must slow sources
  » when available for reuse?
    » Ack or higher level
  - Refuse input when full
    » backpressure in reliable network
    » tree saturation
    » deadlock free
  - what happens to traffic not bound for congested dest?
    » Reserve ack back channel
    » drop packets
  - Utilize higher-level semantics of programming model
Challenges (cont)

- Fetch Deadlock
  - For network to remain deadlock free, nodes must continue accepting messages, even when cannot source msgs
  - What if incoming transaction is a request?
    » Each may generate a response, which cannot be sent!
    » What happens when internal buffering is full?

- logically independent request/reply networks
  - physical networks
  - virtual channels with separate input/output queues
- bound requests and reserve input buffer space
  - K(P-1) requests + K responses per node
  - service discipline to avoid fetch deadlock?

- NACK on input buffer full
  - NACK delivery?

Challenges in Realizing Prog. Models in the Large

- One-way transfer of information
- No global knowledge, nor global control
  - barriers, scans, reduce, global-OR give fuzzy global state
- Very large number of concurrent transactions
- Management of input buffer resources
  - many sources can issue a request and over-commit destination before any see the effect
- Latency is large enough that you are tempted to "take risks"
  - optimistic protocols
  - large transfers
  - dynamic allocation
- Many many more degrees of freedom in design and engineering of these system

Network Transaction Processing

Key Design Issue:
- How much interpretation of the message?
- How much dedicated processing in the Comm. Assist?

Spectrum of Designs

- None: Physical bit stream
  - blind, physical DMA nCUBE, iPSC, ...
- User/System
  - User-level port CM-5, ‘T
  - User-level handler J-Machine, Monsoon, ...
- Remote virtual address
  - Processing, translation Paragon, Meiko CS-2
- Global physical address
  - Proc = Memory controller RP3, BBN, T3D
- Cache-to-cache
  - Cache controller Dash, KSR, Flash
  - Increasing HW Support, Specialization, Intrusiveness, Performance [???]

Shared Physical Address Space

- NI emulates memory controller at source
- NI emulates processor at dest

Case Study: Cray T3D

- Build up info in 'shell'
- Remote memory operations encoded in address
Case Study: NOW

- General purpose processor embedded in NIC

**Context for Scalable Cache Coherence**

- Realizing Pgm Models through net transaction protocols
  - efficient node-to-net interface
  - interprets transactions

- Scalable Networks
  - many simultaneous transactions

- Cache naturally replicates data
  - coherence through bus snooping protocols
  - consistency

- Need cache coherence protocols that scale!
  - no broadcast or single point of order

**Generic Solution: Directories**

- Maintain state vector explicitly
  - associate with memory block
  - records state of block in each cache

- On miss, communicate with directory
  - determine location of cached copies
  - determine action to take
  - conduct protocol to maintain coherence

**A Cache Coherent System Must:**

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (b) Determine when to invoke coherence protocol
    - (a) Find info about state of block in other caches to determine action
    - whether need to communicate with other cached copies
    - (b) Locate the other copies
    - (c) Communicate with those copies (inval/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
- Different approaches distinguished by (a) to (c)

**Adminstrative Break**

- Project Descriptions due today
- Properties of a good project
  - There is an idea
  - There is a body of background work
  - There is something that differentiates the idea
  - There is a reasonable way to evaluate the idea

**Bus-based Coherence**

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a “search”
  - others respond to the search probe and take necessary action

- Could do it in scalable network too
  - broadcast to all processors, and let them respond

- Conceptually simple, but broadcast doesn’t scale with p
  - on bus, bus bandwidth doesn’t scale
  - on scalable network, every fault leads to at least p network transactions

- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol
One Approach: Hierarchical Snooping

- Extend snooping approach: hierarchy of broadcast media
  - tree of buses or rings (KSR-1)
  - processors are in the bus- or ring-based multiprocessors at the leaves
  - parents and children connected by two-way snoopy interfaces
  - snoop both buses and propagate relevant transactions
  - main memory may be centralized at root or distributed among leaves
- Issues (a) - (c) handled similarly to bus, but not full broadcast
  - faulting processor sends out “search” bus transaction on its bus
  - propagates up and down hierarchy based on snoop results

- Problems:
  - high latency: multiple levels, and snoop/lookup at every level
  - bandwidth bottleneck at root
- Not popular today

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

Basic Operation of Directory

- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit
- Read from main memory by processor i:
  - If dirty-bit OFF then { read from main memory; turn p[i] ON; }
  - if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i; }
- Write to main memory by processor i:
  - If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }

Basic Directory Transactions

- P1: pA
- P2: pA
  - Read request to directory
  - Reply with owner identity
  - Read req. to owner
  - Data
  - Reply
  - Revision message to directory

- P1: pA
- P2: pA
  - ReadEx request to directory
  - Reply with sharers identity
  - In val. req. to sharer
  - Inv al. ack

Example Directory Protocol (1st Read)

Example Directory Protocol (Read Share)
Example Directory Protocol (Wr to shared)

Example Directory Protocol (Wr to Ex)

Directory Protocol (other transitions)

A Popular Middle Ground

• Two-level “hierarchy”
• Individual nodes are multiprocessors, connected non-hierarchically
  – e.g. mesh of SMPs
• Coherence across nodes is directory-based
  – directory keeps track of nodes, not individual processors
• Coherence within nodes is snooping or directory
  – orthogonal, but needs a good interface of functionality
• Examples:
  – Convex Exemplar: directory-directory
  – Sequent, Data General, HAL: directory-snoopy
• SMP on a chip?

Example Two-level Hierarchies

Latency Scaling

• $T(n) = \text{Overhead} + \text{Channel Time} + \text{Routing Delay}$
• Overhead?
• Channel Time($n$) = $n/B$ — BW at bottleneck
• RoutingDelay($h,n$)
Typical example

- max distance: \( \log n \)
- number of switches: \( \alpha n \log n \)
- overhead = 1 us, BW = 64 MB/s, 200 ns per hop
- Pipelined
  \( T_{\text{pip}}(128) = 1.0 \text{ us} + 2.0 \text{ us} + 6 \text{ hops } 0.2 \text{ us/hop} = 4.2 \text{ us} \)
  \( T_{\text{pip}}(128) = 1.0 \text{ us} + 2.0 \text{ us} + 10 \text{ hops } 0.2 \text{ us/hop} = 5.0 \text{ us} \)
- Store and Forward
  \( T_{\text{sf}}(128) = 1.0 \text{ us} + 6 \text{ hops } (2.0 + 0.2) \text{ us/hop} = 14.2 \text{ us} \)
  \( T_{\text{sf}}(1024) = 1.0 \text{ us} + 10 \text{ hops } (2.0 + 0.2) \text{ us/hop} = 23 \text{ us} \)

Cost Scaling

- \( \text{cost}(p,m) = \text{fixed cost} + \text{incremental cost}(p,m) \)
- Bus Based SMP?
- Ratio of processors : memory : network : I/O ?
- Parallel efficiency(p) = Speedup(P) / P
- \( \text{Costup}(p) = \text{Cost}(p) / \text{Cost}(1) \)
- Cost-effective: \( \text{speedup}(p) > \text{costup}(p) \)
- Is super-linear speedup possible?