Performance Metrics for Communication Mechanisms

Communication bandwidth - limited by time for which resources within a node are tied up (occupancy), in addition to processor, memory, and interconnection bandwidths

Communication latency - sender overhead + time of flight + transport latency + receiver overhead (overhead and occupancy closely related)

Communication latency hiding - how well can the mechanism hide latency by overlapping communication with computation or other communication

Multiprocessor Systems Design: Coherence, Synchronization, and Consistency Issues

Sandhya Dwarkadas
Department of Computer Science
University of Rochester

Advantages of Shared Memory

Ease of programming when communication patterns are complex or vary dynamically during execution

Better use of bandwidth when communicating small items (lower overhead) - protection implemented in hardware, communication implicit

Use of hardware-controlled automatic caching with reduced latency and contention for accessing shared data

Flynn's Taxonomy of Parallel Architectures

Single instruction stream, single data stream (SISD)
- uniprocessor

Single instruction stream, multiple data streams (SIMD)

Multiple instruction streams, single data stream (MISD)
- no existing commercial implementation

Multiple instruction streams, multiple data streams (MIMD)

Advantages of Message Passing

Simpler hardware - better scalability

Explicit communication, focusing attention on its cost

MIMD Issues

Data sharing - single versus multiple address spaces

Process coordination - synchronization using lock or messages

Distributed versus centralized memory

Connectivity - single shared bus versus network with many different topologies

Fault tolerance

Problems

Amdahl's law - insufficient parallelism

$$Speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

High-latency communication

Portability - knowledge of architecture required

MIMD Machines

Classification based on memory organization
- Centralized shared memory architecture - uniform memory access (UMA)
- distributed memory architectures
  - distributed (or scalable) shared memory architectures (NUMA)
  - Multi-computers

Models for communication
- shared memory
- message passing

Cache Coherence Protocols

Snooping - sharing status broadcast to all copies

Directory-based - sharing status of each block of data kept in one location

Message Passing Machines

Multiple private address spaces - MIMD connected by a network (no lockstep operation)

Interconnection used only for inter-processor communication

Processes communicate via explicit messages (sends and recvs)

Synchronization implicit in the message

Snoopy-Based Protocols

Write Invalidate Protocol

Write Update Protocol

Distributed Shared Memory Without Coherence

Example - Cray T3D

A Simple Invalidate-Based Snoopy Protocol

Finite State Machine with three states -

- Invalid
- Shared/read-only (clean)
- Modified/read-write (not shared)

Single Bus Shared Memory Architecture - MIMD

Mechanisms for Efficiency

Dual cache tags

Multi-level cache hierarchy with inclusion

Coherence Versus Consistency

Cache coherence - ensuring that modifications made by a processor propagate to all (cached) copies of the data - mechanism used

- program order preserved
- Writes to the same location by different processors serialized
- defines the behavior of reads and writes to the same memory location

Consistency - defines when and in what order modifications are propagated to other processors

- defines the behavior of reads and writes with respect to accesses to other memory locations

Implementing Locks Using Coherence

On a SPARC: *ldstub* moves an unsigned byte into the destination register, and rewrites the same byte in memory to all 1s.

```
_Lock:
      ldstub [%o0], %o1
      addcc %g0, %o1, %g0
      bne _Lock
      nop
fin:  jmpl %r15+8, %g0
      nop
_UnLock:
      st %g0, [%o0]
      jmpl %r15+8, %g0
      nop
```

Directory-Based Cache Coherence Protocols

A directory keeps track of the state of every block that may be cached - sharing status in single location

Directory entries distributed along with the memory

States
- Shared
- Uncached
- Exclusive

How would you improve scalability?

Artificially delay processor - exponential backoff

Use a software queue

Example Directory Protocol

Possible messages
- Read miss - request data from home node
- Write miss - request data from home node
- Invalidate - invalidate a shared copy of data
- Fetch - fetch a block from remote cache to home node
- Fetch/invalidate - fetch and invalidate block at remote cache
- Data value reply - return data value from home node
- Data write back - write back data to home node

Hardware Primitives

Queueing locks instead of letting all processors try again

Assumption: directory-based coherence

Synchronization controller returns lock if free on a miss

Else, create a record of the node's request - let's the processor spin on locked value

When freed, controller updates lock variable in selected processor's cache

Basic Hardware Mechanisms for Synchronization

Must provide the ability to atomically read and modify a memory location
- Test-and-set - atomic exchange
- Fetch-and-increment - returns value and atomically increments it
- Load-locked/store conditional - pair of instructions - deduce atomicity if second instruction returns correct value

Memory Consistency Models

When must a processor see a value that has been updated by another processor?

```
P1:   A = 0;              P2:   B = 0;
      ...                       ...
      A = 1;                    B = 1;
L1:   if (B == 0) ...     L2:   if (A == 0) ...
```

Using ll/sc for Atomic Exchange

Swap the content of R4 with the memory location specified by R1

```
try:    mov   R3, R4       ; mov exchange value
        ll    R2, 0(R1)    ; load linked
        sc    R3, 0(R1)    ; store conditional
        beqz  R3, try      ; branch if store fails
        mov   R4, R2       ; put load value in R4
```

Review

A broad overview of multiprocessor systems design issues

- Flynn's classification
- shared memory vs. message passing
- Snoopy-based versus directory-based coherence
- Synchronization primitives
- Consistency models

Consistency Model Classification

Models vary along the following dimensions

- Local order - order of a processor's accesses as seen locally
- Global order - order of a single processor's accesses as seen by each of the other processors
- Interleaved order - order of interleaving of different processor's accesses on other processors

Sequential Consistency

"A system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." [Lamport 79]

Processor Consistency

Also called Total Store Order (TSO - IBM 370, DEC VAX, SPARC)

"A multiprocessor is said to be processor consistent if the result of any execution is the same as if the operations of each individual processor appear in the sequential order specified by its program." [Goodman 91]

Release Consistency

Implemented in DEC Alpha

Assume data-race-free (synchronized) programs [Ghara. 90]

Distinguish ordinary accesses and synchronization accesses

Separate synchronization accesses into acquires and releases

Guarantee consistency of ordinary data only after release $\rightsquigarrow$ architectural optimizations such as buffering to hid latency