Advanced Computer Architecture II:  
Multiprocessor Design

Distributed Memory Multiprocessors III: Basics of Directory Based Coherence

Professor Russ Joseph  
Department of Electrical and Computer Engineering  
Northwestern University

February 8, 2005

Distributed Memory Multiprocessors III:  
Basics of Directory Based Coherence  

Recently: We have talked about ways to build massively parallel, large scale machines.

Today:  
Talk about supporting shared address space programming on distributed memory machines.

Mid-Quarter evaluations after lecture (take 15 minutes or so).

Administrative Stuff

Announcements
From now on, we will meet regularly on Tuesdays only.
Use time to work on projects.
I will be in my office, and we will make one-on-one appointments.
Exam 2 will be on Thursday, 3/3/05

Projects
Should have simulation and tools ready to go,
Don't forget weekly progress reports,

Review: Distributed Memory Machines

We have talked about Non-Uniform Memory Architectures (NUMA),
Today we will talk about cache-coherent Non-Uniform Memory Architectures (cc-NUMA).
Outline
Differences between bus-based and distributed shared memory
Basic design of directory based coherence
Scaling directories
Design alternatives
Summary

Design Goals
Provide hardware/software support for shared address space programming model.
Approach must scale well so that it can be used in very large systems.

On a memory access to a block, we need to:
  Find out about the state of the block in other caches,
  Locate additional copies if needed,
  Communicate with other copies when appropriate.

Distributed Shared Memory vs. Snooping on Bus
Snooping protocols provide correctness: coherence and consistency,
Basic transaction is bus operation,
Bus acts as globally visible broadcast medium.

On distributed machines, broadcasts are expensive,
Basic operation is network transaction,
Only the sender and receiver read (see) the transaction.

Basic Directory Approach
Every virtual address in the system has a home node which contains the only copy in main memory.
Each memory block has a directory entry which tracks copies and state information.
Read and write operations change the state information via network transactions,
Access faults are used to identify reads and writes to relevant blocks.
Aside: Access Faults
Recall page table management in operating system design,
Each page has a page table entry which specifies access rights on page.

Similar concept in directory based caches,
An access that doesn't match protection rights generates an access fault.
The fault is handled by the communication assist.

Terminology
Some definitions that we will need:
- **home node** – where main memory block is allocated
- **dirty node** – where the block has been modified (has current copy)
- **owner node** – where data will be supplied from on a request (either home or dirty node)
- **exclusive node** – the only place where data resides in a cache
- **local node** – where the request for a block originates

State Bits
*Presence bits* are the easiest way to track valid block locations.
Each bit $i$, represents a different node $i$, in the system.
For each block in directory, bit $i$ is set to 1 if node $i$ has a valid copy of that block.
Also track modified copies with a dirty bit.
What can observations can we make about presence and dirty bits?

Read Miss on Clean Block
The read miss triggers the communication assist at local node:
- Read request sent to home node (which has directory).
- Home node sees dirty bit is off, turns on presence bit for requester, and responds with data.
Read Miss on Dirty Block

The read miss triggers the communication assist at local node:
- Read request sent to home node (which has directory).
- Home node sees dirty bit is on, responds with identify of owner (whose presence bit is on).
- Local node sends read request to owner.
- Owner changes its cache state to shared, responds with data and sends revision message to directory (which updates memory and state bits).

Write Miss on Dirty Block

The write miss triggers the communication assist at local node:
- Write request sent to home node (which has directory).
- Home node sees dirty bit is on, responds with identity of owner.
- Local node requests data and invalidation.
- Owner node transfers data and invalidates its own copy.
- Local node updates its cache state and sends revision to directory.

Write Miss on Clean Block

The write miss triggers the communication assist at local node:
- Write request sent to home node (which has directory).
- Home node sees dirty bit is off, clears presence bits and sets dirty bit, responds with data and presence vector.
- Local node sends invalidate request to all nodes with copy.
- Remote nodes send acknowledgements to local node which finally places data in cache in dirty state.
- Owner changes its cache state to shared, responds with data and sends revision message to directory (which updates memory and state bits).

Scaling Issues

The whole reason we are using distributed memory is to allow for much larger systems.
What are problems with the simple presence bit vector?
How many bits are needed in 64 and 256 node systems?

Clearly we need to manage scalability better.
Hierarchical Directories

To this point, we have looked at “flat directories”:

• All the directory information for a memory block kept in a single globally known location.
• This is simple, but what happens if the home node and local node are very far apart?

Hierarchy schemes use logical tree:

  Leaves are processing nodes, internal nodes are directory.
  Messages propagate up and down the tree.

What are the pros and cons of this approach?

Flat, Cache-Based Directories

Home node does not contain identify for all sharers, instead it has a head pointer.
Requests follow doubly linked list to find all nodes with copies of a block.
Often works well since only a few nodes will share a given block.

Flat, Memory-Based Directories

This was our first basic model, but there was a scalability problem...

We can reduce the presence bit overhead by:

• Increasing block size.
• Putting multiple processors in a node visible to directory (e.g. SMP coherence on top of directory coherence).

Another approach is to use limited pointers (an array of sharing nodes).
But we have to address overflow.

Summary

Differences between bus-based and distributed shared memory
Basic design of directory-based coherence
Scaling directories
Design alternatives: flat vs. hierarchical and cache-based vs. memory based

Don’t forget: No class on Thursday!!!