

# Cache Coherence for Large-Scale Machines

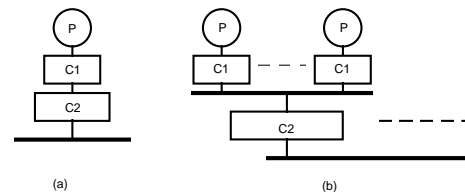
## CS 418

### Lectures 15-16

#### Topics

- Hierarchies
- Directory Protocols

## Hierarchical Cache Coherence

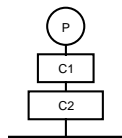


- Hierarchies arise in different ways:
  - (a) A processor with an on-chip and external cache (single cache hierarchy)
  - (b) Large scale multiprocessor using a hierarchy of buses (multi-cache hierarchy)

- 2 -

CS 418 S'04

## Single Cache Hierarchies



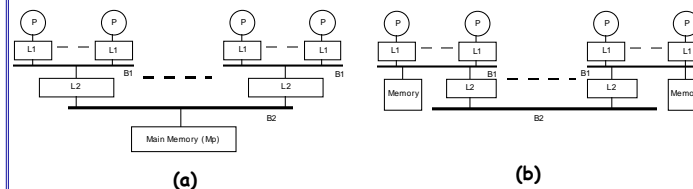
- **Inclusion property:** Everything in L1 cache is also present in L2 cache.
- L2 must also be owner of block if L1 has the block dirty
- Snoop of L2 takes responsibility for recalling or invalidating data due to remote requests
- It often helps if the block size in L1 is smaller or the same size as that in L2 cache

- 3 -

CS 418 S'04

## Hierarchical Snoopy Cache Coherence

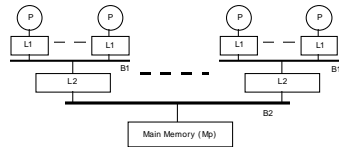
- Simplest way to build large-scale cache-coherent MPs is to use a hierarchy of buses and use snoopy coherence at each level.
- Two possible ways to build such a machine:
  - (a) All main memory at the global (B2) bus
  - (b) Main memory distributed among the clusters



- 4 -

CS 418 S'04

## Hierarchies with Global Memory



- **First-level caches:**
  - Highest performance SRAM caches.
  - B1 follows standard snoopy protocol
- **Second-level caches:**
  - Much larger than L1 caches (set assoc). **Must maintain inclusion.**
  - L2 cache acts as filter for B1-bus and L1-caches.
  - L2 cache can be DRAM based, since fewer references get to it.

- 5 -

CS 418 S'04

## Hierarchies w/ Global Mem (Cont)

### Advantages:

- Misses to main memory just require single traversal to the root of the hierarchy.
- Placement of shared data is not an issue.

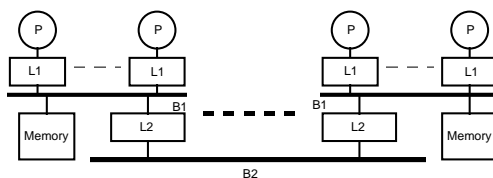
### Disadvantages:

- Misses to local data structures (e.g., stack) also have to traverse the hierarchy, resulting in higher traffic and latency.
- Memory at the global bus must be highly interleaved. Otherwise bandwidth to it will not scale.

- 6 -

CS 418 S'04

## Cluster Based Hierarchies



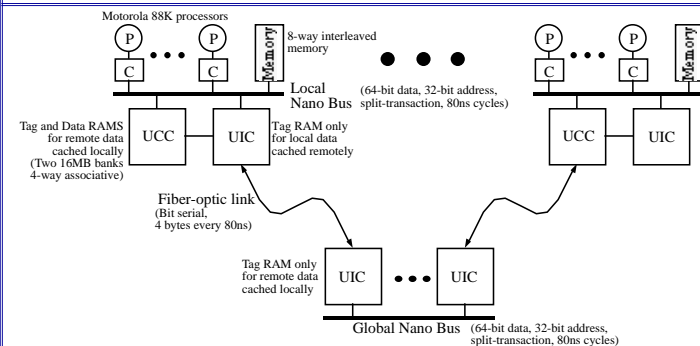
### Key idea: Main memory is distributed among clusters.

- reduces global bus traffic (local data & suitably placed shared data)
- reduces latency (less contention and local accesses are faster)
- example machine: Encore Gigamax
- L2 cache can be replaced by a tag-only router-coherence switch.

- 7 -

CS 418 S'04

## Encore Gigamax



- 8 -

CS 418 S'04

## Cache Coherence in Gigamax

- Write to local-bus is passed to global-bus if:
  - data allocated in remote Mp
  - allocated local but present in some remote cache
- Read to local-bus passed to global-bus if:
  - allocated in remote Mp, and not in cluster cache
  - allocated local but dirty in a remote cache
- Write on global-bus passed to local-bus if:
  - allocated in to local Mp
  - allocated remote, but dirty in local cache
- ...
- Many race conditions possible (e.g., write-back going out as request coming in)

- 9 -

CS 418 S'04

## Hierarchies of Rings (e.g. KSR)

- Hierarchical ring network, not bus
- Snoop on requests passing by on ring
- Point-to-point structure of ring implies:
  - potentially higher bandwidth than buses
  - higher latency

- 10 -

CS 418 S'04

## Hierarchies: Summary

### Advantages:

- Conceptually simple to build (apply snooping recursively)
- Can get merging and combining of requests in hardware

### Disadvantages:

- Physical hierarchies do not provide enough bisection bandwidth (the root becomes a bottleneck, e.g., 2-d, 3-d grid problems)
  - patch solution: multiple buses/rings at higher levels
- Latencies often larger than in direct networks

- 11 -

CS 418 S'04

## Directory Based Cache Coherence

## Motivation for Directory Schemes

Snoopy schemes do not scale because they rely on broadcast

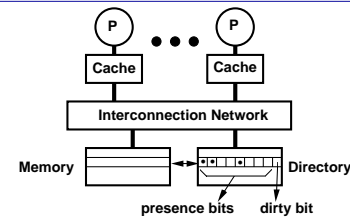
Directory-based schemes allow scaling.

- they avoid broadcasts by keeping track of all PEs caching a memory block, and then using point-to-point messages to maintain coherence
- they allow the flexibility to use any scalable point-to-point network

- 13 -

CS 418 S'04

## Basic Scheme (Censier & Feautrier)



- Assume "k" processors.
- With each cache-block in memory: k presence-bits, and 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

• Read from main memory by PE-i:

- If dirty-bit is OFF then { read from main memory; turn p[i] ON; }
- if dirty-bit is ON then { recall line from dirty PE (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to PE-i; }

• Write to main memory:

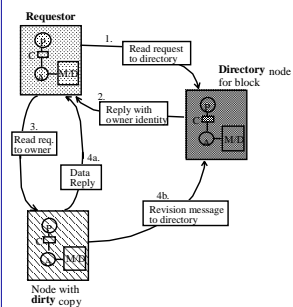
- If dirty-bit OFF then { supply data to PE-i; send invalidations to all PEs caching that block; turn dirty-bit ON; turn P[i] ON; ... }

- 14 -

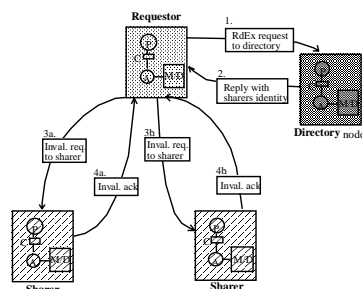
CS 418 S'04

## Directory Protocol Examples

(a) Read miss to a block in dirty state



(b) Write miss to a block with two sharers



Many alternative for organizing directory information

- 15 -

CS 418 S'04

## Scaling with Number of Processors

Scaling of memory and directory bandwidth provided

- Centralized directory is bandwidth bottleneck, just like centralized memory
- How to maintain directory information in distributed way?

Scaling of performance characteristics

- *traffic*: # of network transactions each time protocol is invoked
- *latency*: # of network transactions in critical path each time

Scaling of directory storage requirements

- Number of presence bits needed grows as the number of processors

How directory is organized affects all these, performance at a target scale, as well as coherence management issues

- 16 -

CS 418 S'04

## Insights into Directories

### Inherent program characteristics:

- determine whether directories provide big advantages over broadcast
- provide insights into how to organize and store directory information

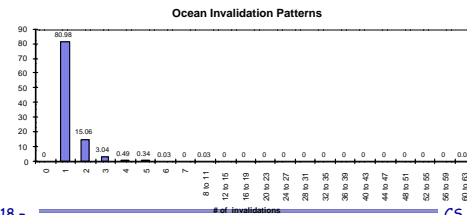
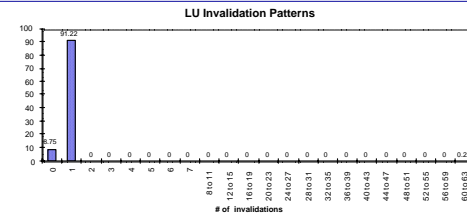
### Characteristics that matter

- frequency of write misses?
- how many sharers on a write miss
- how these scale

- 17 -

CS 418 S'04

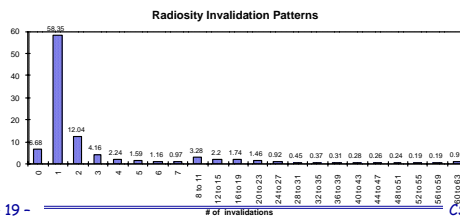
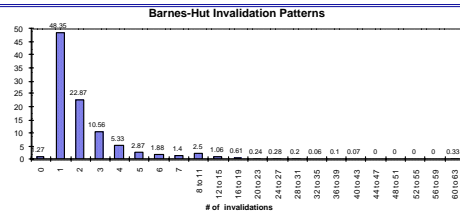
## Cache Invalidation Patterns



- 18 -

CS 418 S'04

## Cache Invalidation Patterns



- 19 -

CS 418 S'04

## Sharing Patterns Summary

Generally, only a few sharers at a write, scales slowly with P

- Code and read-only objects (e.g., scene data in Raytrace)
  - no problems as rarely written
- Migratory objects (e.g., cost array cells in LocusRoute)
  - even as # of PEs scale, only 1-2 invalidations
- Mostly-read objects (e.g., root of tree in Barnes)
  - invalidations are large but infrequent, so little impact on performance
- Frequently read/written objects (e.g., task queues)
  - invalidations usually remain small, though frequent
- Synchronization objects
  - low-contention locks result in small invalidations
  - high-contention locks need special support (SW trees, queueing locks)

Implies directories very useful in containing traffic

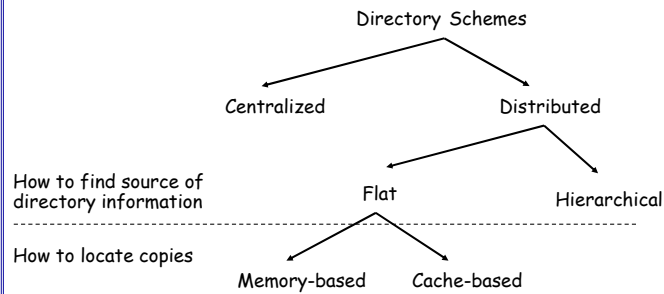
- if organized properly, traffic and latency shouldn't scale too badly

Suggests techniques to reduce storage overhead

- 20 -

CS 418 S'04

## Organizing Directories



Let's see how they work and their scaling characteristics with P

- 21 -

CS 418 S'04

## How to Find Directory Information

**centralized memory and directory - easy: go to it**

- but not scalable

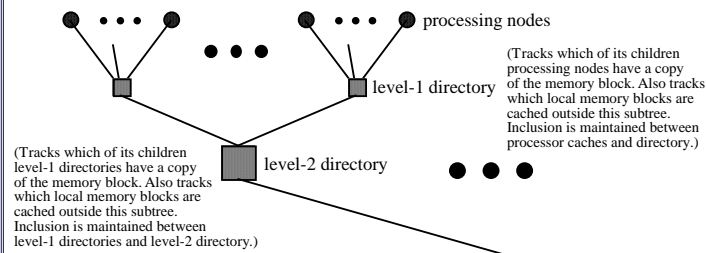
**distributed memory and directory**

- **flat schemes**
  - directory distributed with memory: at the *home*
  - location based on address (hashing): network action sent directly to home
- **hierarchical schemes**
  - directory organized as a hierarchical data structure
  - leaves are processing nodes, internal nodes have only directory state
  - node's directory entry for a block says whether each subtree caches the block
  - to find directory info, send "search" message up to parent
    - » routes itself through directory lookups
  - like hierarchical snooping, but point-to-point messages between children and parents

- 22 -

CS 418 S'04

## How Hierarchical Directories Work



**Directory is a hierarchical data structure**

- leaves are processing nodes, internal nodes just directory
- logical hierarchy, not necessarily physical (can be embedded in general network)

- 23 -

CS 418 S'04

## How Is Location of Copies Stored?

**Hierarchical Schemes**

- through the hierarchy
- each directory has presence bits for its children (subtrees), and dirty bit

**Flat Schemes**

- varies a lot
- different storage overheads and performance characteristics
- **Memory-based schemes**
  - info about copies stored all at the home with the memory block
  - Dash, Alewife, SGI Origin, Flash
- **Cache-based schemes**
  - info about copies distributed among copies themselves
    - » each copy points to next
  - Scalable Coherent Interface (SCI: IEEE standard)

- 24 -

CS 418 S'04

## Flat, Memory-based Schemes

All info about copies colocated with the block itself at the home

- works just like centralized scheme, except physically distributed

### Scaling of performance characteristics

- traffic on a write: proportional to number of sharers
- latency of a write: can issue invalidations to sharers in parallel

- 25 -

CS 418 S'04

## How Does Storage Overhead Scale?

Simplest representation: *full bit vector*

- i.e. one presence bit per node

Directory storage overhead:

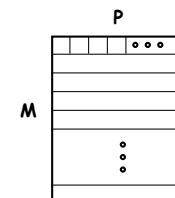
$P$  = # of processors (or nodes)

$M$  = # of blocks in memory

- overhead is proportional to  $P \cdot M$

Does not scale well with  $P$ :

- 64-byte line implies:
  - 64 nodes: 12.7% ovhd.
  - 256 nodes: 50% ovhd.
  - 1024 nodes: 200% ovhd.



- 26 -

CS 418 S'04

## Reducing Storage Overhead

- Full Bit Vector Schemes Revisited

- Limited Pointer Schemes

- reduce "width" of directory (I.e. the "P" term)

- Sparse Directories

- reduce "height" of directory (I.e. the "M" term)

- 27 -

CS 418 S'04

## The Full Bit Vector Scheme

Invalidation traffic is best

- because sharing information is accurate

Optimizations for full bit vector schemes:

- increase cache block size:

- reduces storage overhead proportionally
- problems with this approach?

- use multiprocessor nodes:

- bit per multiprocessor node, not per processor
- still scales as  $P \cdot M$ , but not a problem for all but very large machines
  - » e.g., 256-procs, 4 per cluster, 128B line: 6.25% ovhd.

- 28 -

CS 418 S'04

## Limited Pointer Schemes

### Observation:

- Since data is expected to be in only a few caches at any one time, a limited # of pointers per directory entry should suffice

### Overflow Strategy:

- What to do when # of sharers exceeds # of pointers?

Many different schemes based on differing overflow strategies

- 29 -

CS 418 S'04

## Overflow Schemes for Limited Pointers

### Broadcast ( $Dir_iB$ )

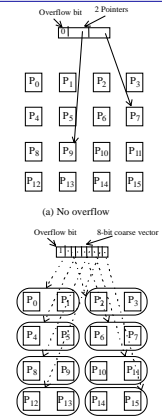
- broadcast bit turned on upon overflow
- when is this bad?

### No-broadcast ( $Dir_iNB$ )

- on overflow, new sharer replaces one of the old ones (invalidated)
- when is this bad?

### Coarse vector ( $Dir_iCV$ )

- change representation to a coarse vector, 1 bit per k nodes
- on a write, invalidate all nodes that a bit corresponds to



- 30 -

CS 418 S'04

## Overflow Schemes (contd.)

### Software ( $Dir_iSW$ )

- trap to software, use any number of pointers (no precision loss)
  - MIT Alewife: 5 ptrs, plus one bit for local node
- but extra cost of interrupt processing on software
  - processor overhead and occupancy
  - latency
    - » 40 to 425 cycles for remote read in Alewife
    - » 84 cycles for 5 inval, 707 for 6.

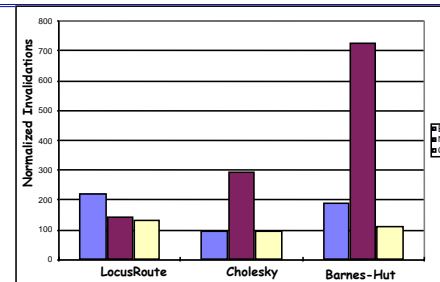
### Dynamic pointers ( $Dir_iDP$ )

- use pointers from a hardware free list in portion of memory
- manipulation done by hw assist, not sw
- e.g. Stanford FLASH

- 31 -

CS 418 S'04

## Some Data



- 64 procs, 4 pointers, normalized to full-bit-vector
- Coarse vector quite robust

### General conclusions:

- full bit vector simple and good for moderate-scale
- several schemes should be fine for large-scale, no clear winner yet

- 32 -

CS 418 S'04



## Reducing Height: Sparse Directories

Reduce  $M$  term in  $P*M$

**Observation:** total number of cache entries  $\ll$  total amount of memory.

- most directory entries are idle most of the time
- 1MB cache and 64MB per node  $\Rightarrow$  98.5% of entries are idle

**Organize directory as a cache**

- but no need for backup store
  - send invalidations to all sharers when entry replaced
- one entry per "line"; no spatial locality
- different access patterns (from many procs, but filtered)
- allows use of SRAM, can be in critical path
- needs high associativity, and should be large enough

Can trade off width and height

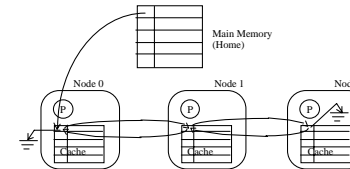
- 33 -

CS 418 S'04

## Flat, Cache-based Schemes

• How they work:

- home only holds pointer to rest of directory info
- distributed linked list of copies, weaves through caches
  - cache tag has pointer, points to next cache with a copy
- on read, add yourself to head of the list (comm. needed)
- on write, propagate chain of invalids down the list



• Scalable Coherent Interface (SCI) IEEE Standard

- doubly linked list

- 34 -

CS 418 S'04

## Scaling Properties (Cache-based)

**Traffic on write:** proportional to number of sharers

**Latency on write:** proportional to number of sharers!

- don't know identity of next sharer until reach current one
- also assist processing at each node along the way
- (even reads involve more than one other assist: home and first sharer on list)

**Storage overhead:** quite good scaling along both axes

- Only one head ptr per memory block
  - rest is all prop to cache size

**Other properties:**

- good: mature, IEEE Standard, fairness
- bad: complex

- 35 -

CS 418 S'04

## Summary of Directory Organizations

**Flat Schemes:**

**Issue (a): finding source of directory data**

- go to home, based on address

**Issue (b): finding out where the copies are**

- memory-based: all info is in directory at home
- cache-based: home has pointer to first element of distributed linked list

**Issue (c): communicating with those copies**

- memory-based: point-to-point messages (perhaps coarser on overflow)
  - can be multicast or overlapped
- cache-based: part of point-to-point linked list traversal to find them
  - serialized

**Hierarchical Schemes:**

- all three issues through sending messages up and down tree
- no single explicit list of sharers
- only direct communication is between parents and children

- 36 -

CS 418 S'04

## Summary of Directory Approaches

**Directories offer scalable coherence on general networks**

- no need for broadcast media

**Many possibilities for organizing directory and managing protocols**

**Hierarchical directories not used much**

- high latency, many network transactions, and bandwidth bottleneck at root

**Both memory-based and cache-based flat schemes are alive**

- for memory-based, full bit vector suffices for moderate scale
  - measured in nodes visible to directory protocol, not processors