Directory-based Cache Coherence:

- What is needed?
- Adv. of MP nodes
- Disadvantages
- Basics of directory
- Directory organization
- Is directory useful?
- Sharing pattern
- Directory organization
- Directory overhead
- Path of a read miss
- Correctness issues

**[From Chapter 8 of Culler, Singh, Gupta]**
**[SGI Origin 2000 material taken from Laudon and Lenoski, ISCA 1997]**
**[GS320 material taken from Gharachorloo et al., ASPLOS 2000]**

Previous Next

## What is needed?

- On every memory operation
  - Find the state of the cache line (normally present in cache)
  - Tell others about the operation if needed (achieved by broadcasting in bus-based or small-scale distributed systems)
  - Other processors must take appropriate actions
  - Also, need a mechanism to resolve races between concurrent conflicting accesses (i.e. one of them is a write): this essentially needs some central control on a per cache line basis
  - Atomic bus provides an easy way of serializing
  - Split-transaction bus with a distributed request table works only because every request table can see every transaction
- Need to have a table that gets accessed/updated by every cache line access
  - This is the directory
  - Every cache line has a separate directory entry
  - The directory entry stores the state of the line, who the current owner is (if any), the sharers (if any), etc.
  - On a miss, the directory entry must be located, and appropriate coherence action must be taken
  - A popular architecture is to have a two-level hierarchy: each node is SMP, kept coherent via snoopy or directory protocol, and nodes are kept coherent by a scalable directory protocol (Convex Exemplar: directory-directory, Sequent, Data General, HAL, DASH: snoopy-directory)

## Adv. of MP nodes

- Amortization of node fixed cost over multiple processors; can use commodity SMPs
- Much communication may be contained within a node i.e. less “remote” communication
- Request combining by some extra hardware in memory controller
- Possible to share caches e.g., chip multiprocessor nodes (IBM POWER4 and POWER5) or hyper-threaded nodes (Intel Xeon MP)
- Exact benefit depends on sharing pattern
  - Widely shared data or nearest neighbor (if properly mapped) may be good

## Disadvantages

- Snoopy bus delays all accesses
  - The local snoop must complete first
  - Then only a request can be sent to remote home
  - Same delay may be incurred at the remote home also depending on the coherence scheme
  - This dictated SGI Origin 2000 to have dual processor nodes, but managed entirely by director
- Bandwidth at critical points is shared by all processors
  - System bus, memory controller, DRAM, router
  - Bad communication patterns can actually result in execution time larger than uniprocessor nodes even though average “hop” time may be larger e.g., compare two

16P systems one with 4-way 4 nodes and one with 16 nodess

Previous Next

## Basics of directory

- Theoretically speaking each directory entry should have a dirty bit and a bitvector of length P
  - On a read from processor k, if dirty bit is off read cache line from memory, send it to k, set bit[k] in vector; if dirty bit is on read owner id from vector (different interpretation of bitvector), send read intervention to owner, owner replies line directly to k (how?), sends a copy to home, home updates memory, directory controller sets bit[k] and bit[owner] in vector
  - On a write from processor k, if dirty bit is off send invalidations to all sharers marked in vector, wait for acknowledgments, read cache line from memory, send it to k, zero out vector and write k in vector, set dirty bit; if dirty bit on same as read, but now intervention is of readX type and memory does not write the line back, dirty bit is set and vector=k

## Directory organization

- Centralized vs. distributed
  - Centralized directory helps to resolve many races, but becomes a bandwidth bottleneck
  - One solution is to provide a banked directory structure: with each memory bank associate its directory bank
  - But since memory is distributed, this essentially leads to distributed directory structure i.e. each node is responsible for holding the directory entries corresponding to the memory lines it is holding
  - Why did we decide to have a distributed memory organization instead of dance hall?

## Is directory useful?

- One drawback of directory
  - Before looking up the directory you cannot decide what to do (even if you start reading memory speculatively)
  - So directory introduces one level of indirection in every request that misses in processor's cache hierarchy
  - Therefore, broadcast is definitely preferable over directory if the system can offer enough memory controller and router bandwidth to handle broadcast messages (network link bandwidth is normally not the bottleneck since most messages do not carry data; observe that you would never broadcast a reply); AMD Opteron adopted this scheme, but target is small scale
- Directory is preferable
  - If number of sharers is small because in this case a broadcast would waste enormous amount of memory controller bandwidth

Previous Next

## Sharing pattern

- Problem is with the writes
  - Frequently written cache lines exhibit a small number of sharers; so small number of invalidations
  - Widely shared data are written infrequently; so large number of invalidations, but rare
  - Synchronization variables are notorious: heavily contended locks are widely shared and written in quick succession generating a burst of invalidations; require special solutions such as queue locks or tree barriers
  - What about interventions? These are very problematic because in these cases you cannot send the interventions before looking up the directory and any speculative memory lookup would be useless
  - For scientific applications interventions are small due to mostly one producer-many consumer pattern; for database workloads these take the lion's share due to migratory pattern and tend to increase with bigger cache
- Optimizing interventions related to migratory sharing has been a major focus of high-end scalable servers
  - AlphaServer GS320 employs few optimizations to quickly resolve races related to migratory hand-off (more later)
  - Some academic research looked at destination or owner prediction to speculatively send interventions even before consulting the directory (Martin and Hill 2003, Acacio et al 2002)
- In general, directory provides far better utilization of bandwidth for scalable MPs compared to broadcast

## Directory organization

- How to find source of directory information
  - Centralized: just access the directory (bandwidth limited)
  - Distributed: flat scheme distributes directory with memory and every cache line has a home node where its memory and directory reside
  - Hierarchical scheme organizes the processors as the leaves of a logical tree (need not be binary) and an internal node stores the directory entries for the memory lines local to its children; a directory entry essentially tells you which of its children subtrees are caching the line and if some subtree which is not its children is also caching; finding the directory entry of a cache line involves a traversal of the tree until the entry is found (inclusion is maintained between level k and k+1 directory node where the root is at the highest level i.e. in the worst case may have to go to the root to find dir.)
- Format of a directory entry
  - Varies a lot: no specific rule
  - Memory-based scheme: directory entry is co-located in the home node with the memory line; various organizations can be used; the most popular one is a simple bit vector (with a 128 bytes line, storage overhead for 64 nodes is 6.35%, for 256 nodes 25%, for 1024 nodes 100%); clearly does not scale with P (more later)
  - Cache-based scheme: Organize the directory as a distributed linked-list where the sharer nodes form a chain; the cache tag is extended to hold a node number; the home node only knows the id of the first sharer; on a read miss the requester adds itself to the head (involves home and first sharer); on a write miss traverse list and

invalidate (essentially serialized chain of messages); advantage: distributes contention and does not make the home node a hot-spot, storage overhead is fixed; but very complex (IEEE SCI standard)

- Lot of research has been done to reduce directory storage overhead
  - The trade-off is between preciseness of information and performance
  - Normal trick is to have a superset of information e.g., group every two sharers into a cluster and have a bit per cluster: may lead to one useless invalidation per cluster
  - We will explore this in detail later
  - Memory-based bitvector scheme is very popular: invalidations can be overlapped or multicast
  - Cache-based schemes incur serialized message chain for invalidation
  - Hierarchical schemes are not used much due to high latency and volume of messages (up and down tree); also root may become a bandwidth bottleneck

Previous Next

## Directory overhead

- Quadratic in number of processors for bitvector
  - Assume P processors, each with M amount of local memory (i.e. total shared memory size is M*P)
  - Let the coherence granularity (cache block size) be B
  - Number of cache blocks per node = M/B = number of directory entries per node
  - Size of one directory entry = P + O(1)
  - Total size of directory memory across all processors = $(M/B)(P+O(1))*P = O(P^2)$

## Path of a read miss

- Assume that the line is not shared by anyone
  - Load issues from load queue (for data) or fetcher accesses icache; looks up TLB and gets PA
  - Misses in L1, L2, L3,… caches
  - Launches address and request type on system bus
  - The request gets queued in memory controller and registered in OTT or TTT (Outstanding Transaction Table or Transactions in Transit Table)
  - Memory controller eventually schedules the request
  - Decodes home node from upper few bits of address
  - Local home: access directory and data memory (how?)
  - Remote home: request gets queued in network interface
- From NI onward
  - Eventually the request gets forwarded to the router and through the network to the home
  - At the home the request gets queued in NI and waits for being scheduled by the home memory controller
  - After it is scheduled home memory controller looks up directory and data memory
  - Reply returns through the same path
- Total time (by log model and memory latency m)
  - Local home: $\max(k_h o, m)$
  - Remote home: $k_r o + g_{h+a} + NI + g_{h+a} + \max(k_h o, m) + g_{h+a+d} + NI + g_{h+a+d} + k_r o$

## Correctness issues

- Serialization to a location
  - Schedule order at home
  - Use NACKs (extra traffic and livelock) or smarter techniques (back-off, NACK-free)
- Flow control deadlock
  - Avoid buffer dependence cycles
  - Avoid network queue dependence cycles
  - Virtual networks multiplexed on physical networks
  - Coherence protocol dictates the virtual network usage