Multiprocessors II: CC-NUMA DSM

Today's topics:

DSM cache coherence

- the hardware stuff
- what happens when we lose snooping
- new issues: global vs. local cache line state
  - enter the directory
- issues of increasing the physical extent of the system

Synchronization basics

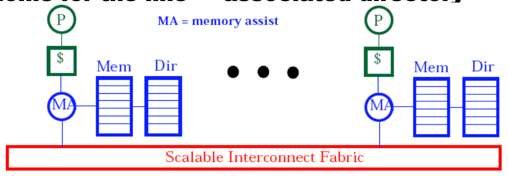
- the software stuff

CC-NUMA for Large Systems

- Bus clearly doesn't scale
  - so replace it with a scalable interconnect
  - similar goals
    - » shared memory and the same ease of use issues
  - similar problems
    - » coherence and consistency
- New problems
  - lose bus as the atomicity/"decider" point
    - » need to replace it with something else
      - directory
  - multi-path communication
    - » longer delays and potential network transaction reordering
  - main memory is now distributed – e.g. DSM
    - » need to figure out where to look for the appropriate copy
  - deadlock and livelock
    - » by product of interconnection network and distributed environment

Directory Concept

- **Atomicity**
  - bus becomes a piece of memory somewhere
    - » a.k.a. a directory
  - coherence entity is still a cache line
    - » could be a page but false sharing would be a problem
  - new state model
    - » local state: kept in the cache holding a copy
    - » global state: kept in the directory
      - who is sharing and in what form
- **Flat main-memory**
  - address high-order bits specify location
    - » "home for the line" – associated directory



Memory Assist

- **2 transaction types**
  - directory vs. memory
    - » shared memory reference ➔ directory
    - » non-shared local reference ➔ normal memory access
      - easy to expand this concept to private but external memory access
        - – again by page
- **Local vs. external reference main memory access**
  - both go through the associated memory controller
  - remote accesses
    - » local remote memory controller
      - for me or not
        - – if not send an access "message" to the appropriate controller
    - » note strange coupling
      - unified memory but with an interconnect
      - both memory and interconnect get involved

CC-NUMA/DSM Conceptual Change

- Unified main-memory address space
  - spread over N physical locations
    - » each with associated memory controller
  - terminology
    - » home node – place where the memory copy exists
    - » local node – originator of the memory request
    - » remote nodes – places where line copies are stored in a cache
- Temporal issues
  - transactions now may need to be remote
    - » even worse – may involve several distributed sites
    - » increased delay creates a problem
      - illusion of coherency gets a bit harder to reliably control
- Interconnect load
  - potential for lots of “coherence” messages
    - » increased contention, power, and delay

Local vs. Global State

- Local
  - similar to SMP model
    - » MSI, MESI, … (lots of options)
- Global – distributed → new issue
  - write-invalidate example
    - » uncached – no processor has a copy so get it from main mem
    - » shared
      - keep list of sharers – memory still has a valid copy
    - » exclusive
      - node X has the only valid copy
        - – memory may be inconsistent

Implementing DSM

- **Atomicity**
  - no longer observed by all since snooping doesn't work
  - atomicity point is now the directory
    - » all nodes need to know about this order
    - » hence messages sent to participant
      - question is how
- **2 interacting FSM's w/ additional delay**
  - global @ home node
  - local @ caches holding a shared copy
  - delay complications
    - » some requests may not succeed
      - multi-person group analogy
        - req: "hey I need a piece of your time @ X
        - response: "Nope I can't do it"
      - several issues
        - requests can be pending
        - or they can be instantly ACK'd or NACK'd by the directory

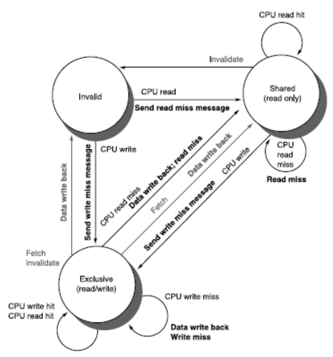
Message Types

- **Write-Invalidate protocol example**
  - WI is common but WU possible
    - » suggestion: think about the differences

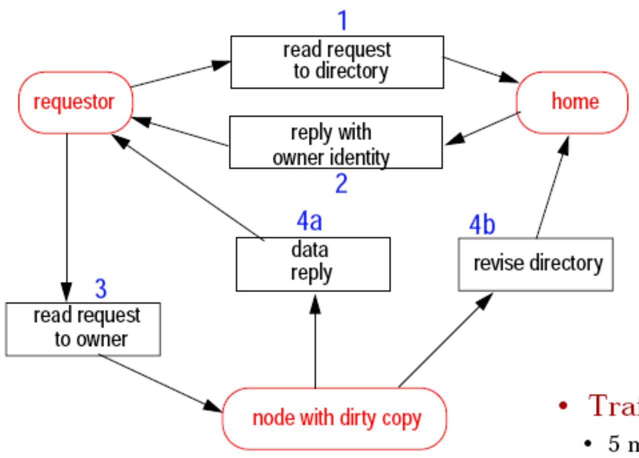
Type	SRC	DEST	Contents	Function
RdMiss	Local $	Home Dir	P, A	Processor P has a read miss at address A, make P a sharer
WrMiss	Local $	Home Dir	P, A	P write misses at A, make P exclusive owner
Invalidate	Home Dir	Remote $	A	invaliidate A
Fetch	Home Dir	Remote $	A	Fetch line A, change local state to shared
Fetch/Inv	Home Dir	Remote $	A	Fetch line A, invalidate line A
Data Reply	Home Dir	Local $	D	Here's the line you requested (Rd or Wr. Miss)
Data WrBk	Remote $	Home Dir	A, D	Update main memory

Local FSM

- **Sources**
  - local processor requests in bold
  - directory requests in normal font
  - MSI protocol (E=M in this FSM)



More Detail: Read Miss to M block



Cache to Cache Copy Style

Any races?

What's missing?

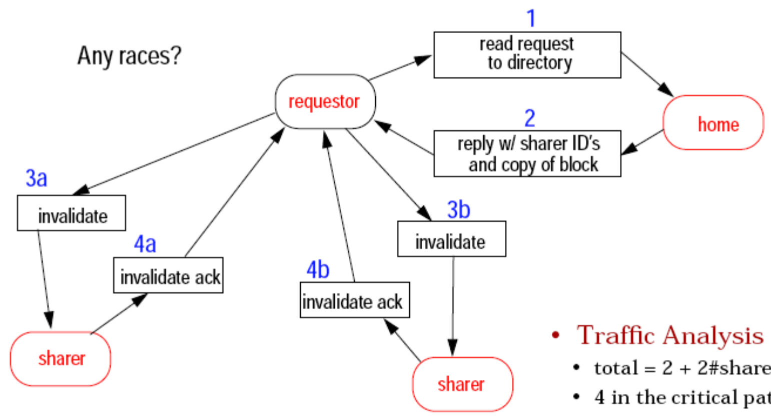
- Traffic Analysis
  - 5 messages total
  - 4 in the critical path

Read Miss Analysis

- **Races**
  - **no problem for single requester**
  - **4a and 4b is non-critical if directory does the right thing**
- **Missing**
  - **concurrent requests for same block**
    - » **ordering at the directory**
      - **one gets seen as first – others?**
        - – **what do you do**
          - – **queue until 4b or NAC 2nd request**
          - – **NACK is easiest**

School of Computing
University of Utah
11
CS6810

Write Miss to Clean Copy

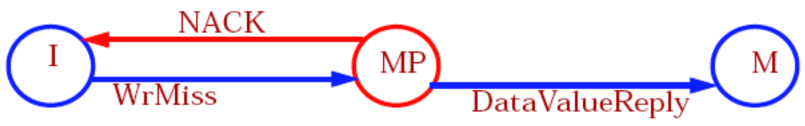


- Traffic Analysis
  - total = 2 + 2#sharers
  - 4 in the critical path

School of Computing
University of Utah
12
CS6810

Additional Considerations

- **Local node may also be home**
  - **MA duties**
    - » shared and local → message to the directory
    - » private → normal memory access
      - either local or remote
      - overload MA
        - – handles inter- and intra-node traffic
        - – priority?
- **NACKS**
  - **directory needs to keep track of pending transactions**

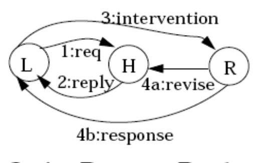


Scalability

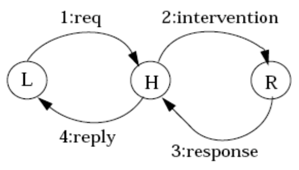
- **Performance**
  - **lots of coherence messages**
    - » contention in interconnect causes additional delay
      - more on this later
- **Meta-data (directory) = directory size**
  - **full map – 1 bit per sharer held in directory**
    - » 64 P's and 64B line
      - 8 directory bytes = 12.5%
    - » 256 P's = 50% OUCH!
    - » 1024 P's = 200% UNACCEPTABLE
  - **other options**
    - » lots of sharers are rare
      - keep small-num ID's
        - – pointer to extended list
      - distributed linked list
        - – too much delay and traffic

Protocol Optimization Example

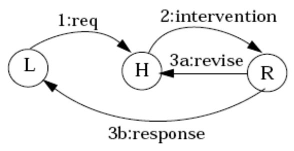
- **Remote node has an exclusive copy**
  - **flat main memory**



Strict Request-Reply



Intervention Forwarding



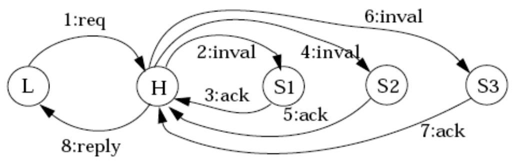
Reply Forwarding

Issues:

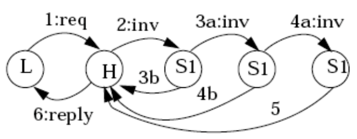
- bottleneck possibilities
- deadlock avoidance

Optimizations II

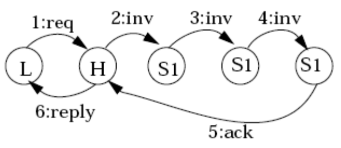
- **Write miss to 3 sharers***



Strict Request Reply

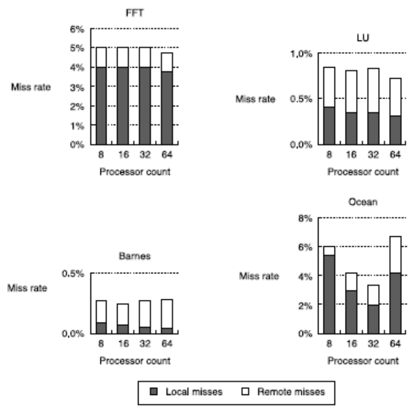


All sharers ACK



One sharer ACK

Miss Rate vs. Processor Count



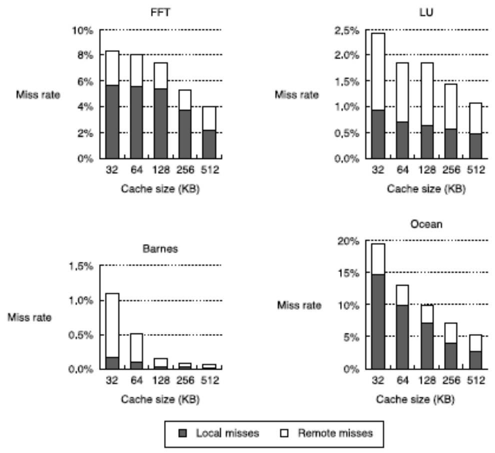
Surprisingly flat

Ocean exhibits what you'd expect to be more generally true

e.g. remote miss rate goes up with PE count

& local miss rate goes down or up depending on how the blocking affects the working set

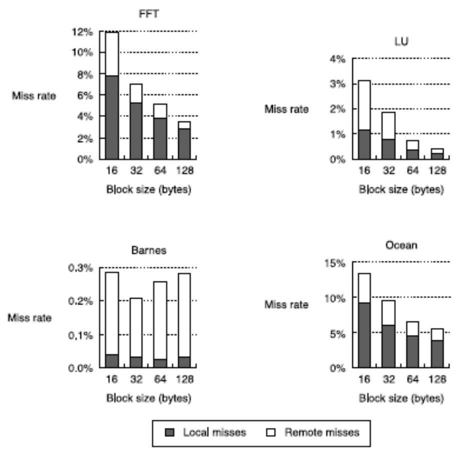
Miss Rate vs. Cache Size



decline as expected

remote miss effect varies - if remote miss was due to capacity then it is reduced - if a true communication miss then it's not affected

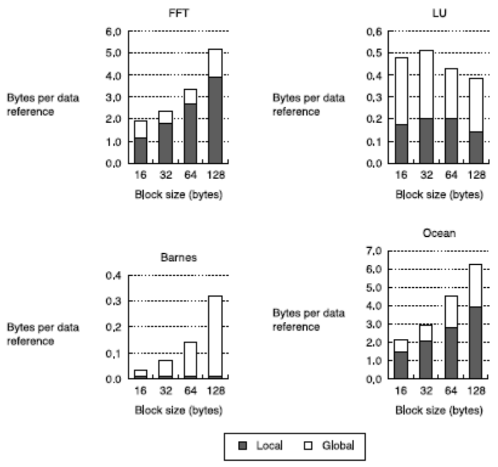
Miss Rate vs. Block Size



as with SMP Barnes is affected by false sharing increase caused by the larger block size

the others benefit due to improved spatial locality

Memory Pressure vs. Block Size



Shows high miss rate of FFT and Ocean

False sharing problem in Barnes

Low miss rate and low false sharing in LU

Latency ala SGI Origin O3K

- 500 MHz CPU
  - model lacks contention and occupancy effects
    - » e.g. measured unloaded – hence best case
    - » DSM has striking impact on communication load

Characteristic	CPU clocks <= 16 PE's	CPU clocks 17-64 PE's
Cache Hit	1	1
Miss to Local Memory	85	85
Remote Miss served by Home	125	150
Remote Miss served by Remote $ (known as a 3-hop miss)	140	170

DSM Coherence vs. Synchronization

- DSM coherence
  - assuming the programmer got it right
    - » DSM coherence keeps the shared memory illusion alive
- Real concurrency has potential problems
  - concurrent writers is the main issue
    - » other hazards also apply RAW, Wax
  - hence synchronization is also needed
    - » locks or semaphores are the norm
    - » who's responsible
      - ultimately some burden is on the programmer
      - system software or hardware support may provide better API's

Synchronization

- 3 components for lock support
  - acquire method
  - waiting method if acquire fails
    - » spin/busy wait: problem is occupancy
      - consumes power and displaces other processing
    - » blocking wait: support of wake-up is needed
    - » which is better?
      - depends on wait time
        - short then spin
        - long then block
        - how do you know?
  - a release method
- Obvious goal
  - mutual exclusion for the critical section
    - » what's the key to getting this right?
- Are locks enough?
  - hint: consider blocked MATMUL, Ocean, …

Lock Atomicity

- Hardware
  - seen in older machines like Cray XMP
    - » lock registers
      - global resource but located in one atomic place
        - get there first and win
    - » bus lock lines in BSP
      - bus arbiter is the atomicity point
- Memory location
  - must be shared
    - » same atomicity issue as CC-DSM
    - » what's the issue here?
- Key
  - hardware must provide some atomicity support
    - » bus arbitration
    - » directory & interconnection network

Hardware Atomicity Support

- Read-Modify-Write
  - more of an interconnect property than an instruction set issue
    - » bus
      - hold bus after read, modify if unlocked, release bus
      - problems
        - – holding bus increases bus bottleneck likelihood
    - » distributed interconnect
      - ideas?
- Test & Set instruction
  - read value but set location to 1 (1➔ locked)
    - » less occupancy than RMW
    - » if return value is 1 then previous lock exists so try again
  - other variants
    - » swap: exchange register value with a memory location

Lock Algorithm Goals

- Low latency
  - quick when not contention
  - slower if there’s competition – fact of life
- Low occupancy
  - minimize amount of interconnect traffic
- Scalability
  - no worse than linear traffic and latency increase
- Low storage overhead
  - minimize meta-data
- Fairness
  - hard to be truly fair
  - redefine as starvation free
    - » e.g. guarantee that requester will never wait forever

DSM Locks

- e.g. SGI LL & SC instructions
  - Load locked and store conditional
    - » LL loads the shared synchronization variable or lock
    - » SC writes it back if no intervening invalidate
      - SC success is indicated by a condition flag
        - – fail if write to invalid line
        - – succeed if write to valid line
    - » What's the problem?

DSM Locks

- e.g. SGI LL & SC instructions
  - Load locked and store conditional
    - » LL loads the shared synchronization variable or lock
    - » SC writes it back if no intervening invalidate
      - SC success is indicated by a condition flag
        - – fail if write to invalid line
        - – succeed if write to valid line
    - » What's the problem?
      - 2 kinds of invalidates
        - – DSM invalidate works just fine
        - – victimized invalidate – oops
          - – replacement policy
            - – never invalidate a LL line
              - – requires a tag bit
            - – mark victimized invalidate
              - – requires a tag bit

More Advanced Goals

- Reduced contention?
  - need some back-off model to desynchronize
    - » e.g. ethernet exponentional back-off idea
- On lock release
  - #1 have only one waiting process try for the lock
    - » also reduces contention
  - #2 have only one waiting process incur a read miss
- Enter more advanced protocols
  - ticket lock does #1
  - array based lock does both
  - both are fair in that they effectively create a FIFO grant order

Ticket Lock

- Take a number
  - each process reads lock and gets next number
    - » from a number serving variable
    - » next requester invalidates you but you have your number
  - read the “now serving” variable
    - » normal reads so no invalidation until the number changes
    - » read your number then go
  - release
    - » update the now-serving number
    - » “fetch & increment”
      - one instance of fetch & op hardware support
  - optimization
    - » delay next read based on difference between
      - your number and “now serving” variable

Array Based Lock

- Get location rather than value
  - p processes/threads → p locations
    - » essentially a queue
      - pad array to get one location per cache line
        - – reduced invalidations
        - – increased storage overhead
          - – good until p gets huge
    - » when your location goes to 1 you get the lock
      - release sets next owners location to 1
      - change in shared line value invalidates next owner's line
- What happens when lock request is in a loop
  - locations may wrap around
    - » sense bit is shared
      - all 1's were written so now it's time to look for a 0 as success

Barrier's

- The other useful synchronization
  - all participants must arrive before any can leave
  - useful for phase exchange
    - » internal and then update boundary values
    - » exchange boundary values
      - e.g. Ocean
- HW support
  - fetch and decrement
    - » n participating – initialize barrier to n-1
      - participant arrival – decrement barrier variable
    - » problem – every arrival invalidates all lines
      - improvement – barrier monitor
        - – sets a second line value indicating arrival

Implications for Software

- Computation phases
  - maximize locality and minimize interconnect traffic
- Data allocation
  - pad to
    - » minimize false sharing & align on cache line boundary
    - » particularly important for arrays
- Conflict misses
  - keep data set for a phase in non-conflicting locations
- Minimize delay
  - may involve extra copies to keep a private local version
  - may involve recomputing
    - » if cheaper than getting value remotely

Concluding Remarks

- For large parallel machines
  - DSM may have become extinct
    - » SUN was the last to go
    - » IBM, Cray, HP, Dell, ... move to message passing
      - hardware is simplified and power is saved
- For small parallel machines
  - e.g. multi-core chips
    - » idea may still play well
- Where is the inflection point
  - it's all about delay and energy cost
    - » long wires are the culprit
  - answer
    - » something to ponder
    - » nobody has provided a definitive model