CS184b: Computer Architecture (Abstractions and Optimizations)

Day 21: May 18, 2005
Shared Memory

Today

- Shared Memory
  - Model
  - Bus-based Snooping
  - Cache Coherence
  - Distributed Shared Memory

Shared Memory Model

- Same model as multithreaded uniprocessor
  - Single, shared, global address space
  - Multiple threads (PCs)
  - Run in same address space
  - Communicate through memory
    - Memory appear identical between threads
    - Hidden from users (looks like memory op)

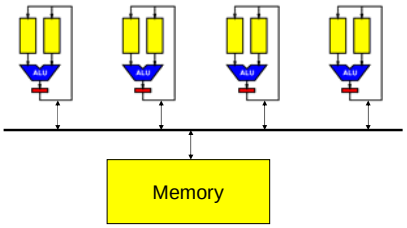
Synchronization

- For correctness have to worry about synchronization
  - Otherwise non-deterministic behavior
  - Threads run asynchronously
  - Without additional/synchronization discipline
    - Cannot say anything about relative timing
- Subject of Friday's Lecture

Models

- **Conceptual model**:
  - Processor per thread
  - Single shared memory
- **Programming Model**:
  - Sequential language
  - Thread package
  - Synchronization primitives
- **Architecture Model**: Multithreaded uniprocessor

Conceptual Model



Architecture Model Implications

- Coherent view of memory
  - Any processor reading at time X will see **same** value
  - All writes eventually effect memory
    - Until overwritten
  - Writes to memory seen in same order by all processors
- Sequentially Consistent Memory View

Sequential Consistency

- Memory must reflect some valid sequential interleaving of the threads

Sequential Consistency

- P1: A = 0
- P2: B = 0
-
- A = 1
- B = 1
- L1: if (B==0)
- L2: if (A==0)

Can both conditionals be true?

Sequential Consistency

- P1: A = 0
- P2: B = 0
-
- A = 1
- B = 1
- L1: if (B==0)
- L2: if (A==0)

Both can be false

Sequential Consistency

- P1: A = 0
- P2: B = 0
-
- A = 1
- B = 1
- L1: if (B==0)
- L2: if (A==0)

If enter L1, then A must be 1
→ not enter L2

Sequential Consistency

- P1: A = 0
- P2: B = 0
-
- A = 1
- B = 1
- L1: if (B==0)
- L2: if (A==0)

If enter L2, then B must be 1
→ not enter L1

Coherence Alone

- Coherent view of memory
  - Any processor reading at time X will see same value
  - All writes eventually effect memory
    - Until overwritten
  - Writes to memory seen in same order by all processors
- Coherence alone does not guarantee sequential consistency

Sequential Consistency

- P1: A = 0
-
- A = 1
- L1: if (B==0)

- P2: B = 0
- B = 1
- L2: if (A==0)

If not force visible changes of variable, (assignments of A, B), could end up inside both.

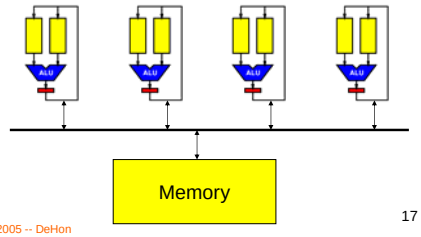
Consistency

- Deals with when written value must be seen by readers
- **Coherence** – w/ respect to same memory location
- **Consistency** – w/ respect to other memory locations
- …there are less strict consistency models…

Implementation

Naïve

- What's wrong with naïve model?



What's Wrong?

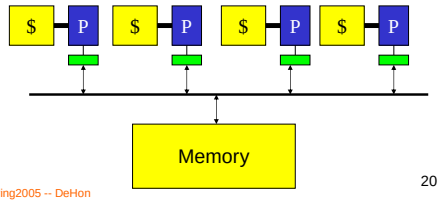
- Memory bandwidth
  - 1 instruction reference per instruction
  - 0.3 memory references per instruction
  - 333ps cycle
  - N*5 Gwords/s ?
- Interconnect
- Memory access latency

Optimizing

- How do we improve?

Naïve Caching

- What happens when add caches to processors?



Naïve Caching

- Cached answers may be stale
- Shadow the correct value

How have both?

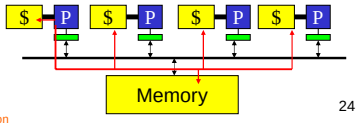
- Keep caching
  - Reduces main memory bandwidth
  - Reduces access latency
- Satisfy Model

Cache Coherence

- Make sure everyone sees **same** values
- Avoid having stale values in caches
- At end of write, all cached values should be the same

Idea

- Make sure everyone sees the new value
- Broadcast new value to everyone who needs it
  - Use bus in shared-bus system



Effects

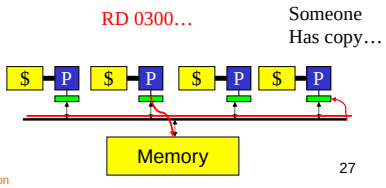
- Memory traffic is now just:
  - Cache misses
  - All writes

Additional Structure?

- Only necessary to write/broadcast a value if someone else has it cached
- Can write locally if know sole owner
  - Reduces main memory traffic
  - Reduces write latency

Idea

- Track usage in cache state
- "Snoop" on shared bus to detect changes in state



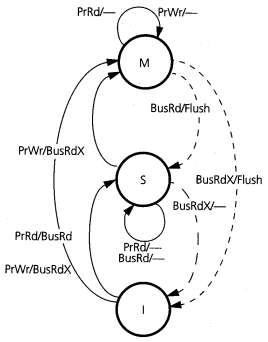
Cache State

- Data in cache can be in one of several states
  - Not cached (not present)
  - Exclusive (not shared)
    - Safe to write to
  - Shared
    - Must share writes with others
- Update state with each memory op

Cache Protocol

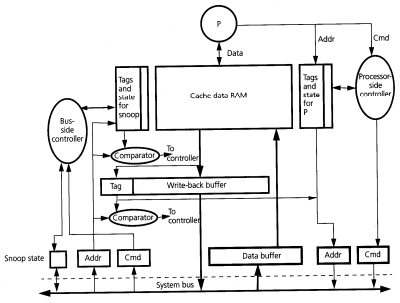
RdX = Read Exclusive

Perform Write by:
•Reading exclusive
•Writing locally



[Culler/Singh/Gupta 5.13]

Snoopy Cache Organization

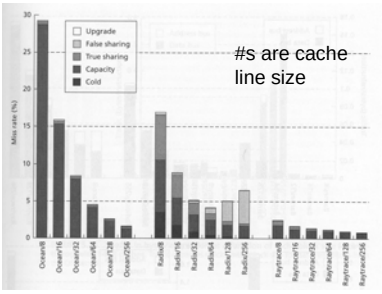


[Culler/Singh/Gupta 6.4]

Cache States

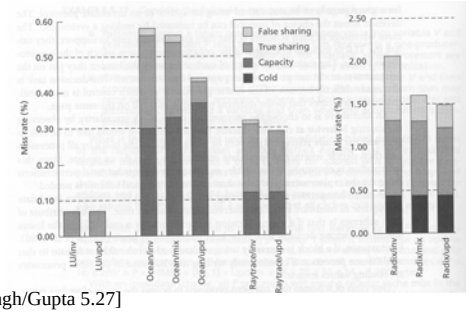
- Extra bits in cache
  - Like valid, dirty

Misses



[Culler/Singh/Gupta 5.23]

Misses

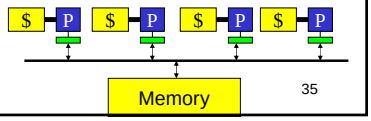


[Culler/Singh/Gupta 5.27]

Distributed Shared Memory

Review

- Shared Memory
  - Programming Model
  - Architectural Model
  - Shared-Bus Implementation
  - Caching Possible w/ Care for Coherence



Previously

- Message Passing
  - Minimal concurrency model
  - Admits general network (not just bus)
  - Messaging overheads and optimization

Last Half

- Distributed Shared Memory
  - No broadcast
  - Memory distributed among nodes
  - Directory Schemes
  - Built on Message Passing Primitives

Snoop Cache Review

- Why did we need broadcast in Snoop-Bus protocol?

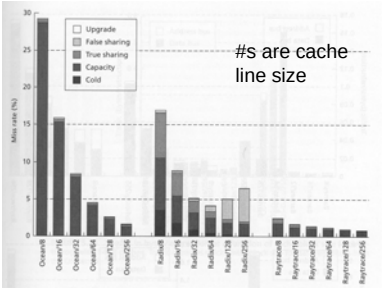
Snoop Cache

- Why did we need broadcast in Snoop-Bus protocol?
  - Detect sharing
  - Get authoritative answer when dirty

Scalability Problem?

- Why can't we use Snoop protocol with more general/scalable network?
  - Mesh
  - fat-tree
  - multistage network
- Single memory bottleneck?

Misses



[Culler/Singh/Gupta 5.23]

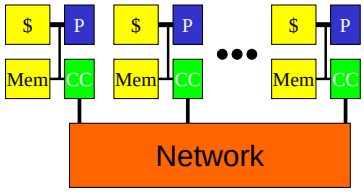
Sub Problems

- How does exclusive owner know when sharing created?
- How know every user?
  - know who needs invalidation?
- How find authoritative copy?
  - when dirty and cached?

Distributed Memory

- Could use Banking to provide memory bandwidth
  - have network between processor nodes and memory banks
- …But, already need network connecting processors
- Unify interconnect and modules
  - each node gets piece of "main" memory

Distributed Memory



"Directory" Solution

- Main memory keeps track of users of memory location
- Main memory acts as **rendezvous** point
- On write,
  - inform all users
    - only need to inform users, not everyone
- On dirty read,
  - forward read request to owner

Directory

- Initial Ideal
  - main memory/home location knows
    - state (shared, exclusive, unused)
    - all sharers

Directory Behavior

- On read:
  - unused
    - give (exclusive) copy to requester
    - record owner
  - (exclusive) shared
    - (send share message to current exclusive owner)
    - record user
    - return value

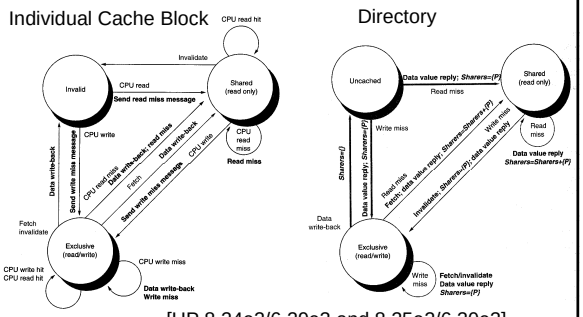
Directory Behavior

- On read:
  - exclusive dirty
    - forward read request to exclusive owner

Directory Behavior

- On Write
  - send invalidate messages to all hosts caching values
- On Write-Thru/Write-back
  - update value

Directory



[HP 8.24e2/6.29e3 and 8.25e2/6.30e3]

Representation

- How do we keep track of readers (owner) ?
  - Represent
  - Manage in Memory

Directory Representation

- Simple:
  - bit vector of readers
  - scalability?
    - State requirements scale as square of number of processors
    - Have to pick maximum number of processors when committing hardware design

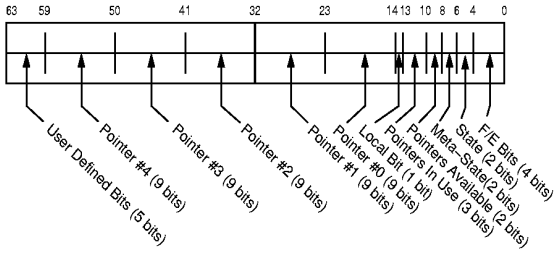
Directory Representation

- Limited:
  - Only allow a small (constant) number of readers
  - Force invalidation to keep down
  - Common case: little sharing
  - weakness:
    - yield thrashing/excessive traffic on heavily shared locations
      - *e.g.* synchronization variables

Directory Representation

- LimitLESS
  - Common case: small number sharing in hardware
  - Overflow bit
  - Store additional sharers in central memory
  - Trap to software to handle
  - TLB-like solution
    - common case in hardware
    - software trap/assist for rest

Alewife Directory Entry



[Agarwal et. al. ISCA'95]

Alewife Timings

Miss Type	Home Location	# Inv. Msgs	hw/ sw	Miss Penalty Cycles	Miss Penalty μsec
Load	local	0	hw	11	0.55
	remote	0	hw	38	1.90
	remote (2-party)	1	hw	42	2.10
	remote (3-party)	1	hw	63	3.15
	remote	–	sw†	425	21.25
Store	local	0	hw	12	0.60
	local	1	hw	40	2.00
	remote	0	hw	38	1.90
	remote (2-party)	1	hw	43	2.15
	remote (3-party)	1	hw	66	3.30
	remote	5	hw	84	4.20
	remote	6	sw	707	35.35

† This sw read time represents the throughput seen by a single node that invokes LimitLESS handling at a sw-limited rate.

[Agarwal et. al. ISCA'95]

Alewife Nearest Neighbor Remote Access Cycles

Action	Count
Cache-miss to request in network	2
Request transit time (8 bytes)	7
Request at memory to output header transmit	7
Data return in network (24 bytes)	15
Response arrival to beginning of cache fill	3
Cache fill time	4

[Agarwal et. al. ISCA'95]

Alewife Performance

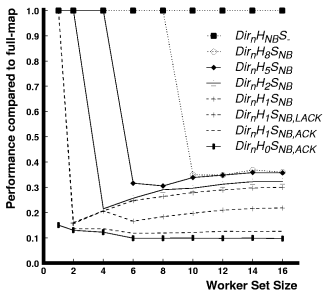
Program	Running Time (Mcycles) 1P	2P	4P	8P	16P	32P	Speedup 1P	2P	4P	8P	16P	32P
Orig MP3D	67.6	41.7	24.8	13.9	7.4	4.3	1.0	1.6	2.7	4.9	9.2	15.7
Mod MP3D	47.4	24.5	12.4	6.9	3.5	2.2	1.0	1.9	3.8	6.9	13.4	21.9
Barnes-Hut	9144.6	4776.5	2486.9	1319.4	719.6	434.2	1.0	1.9	3.7	6.9	12.7	21.1
Barnes-Hut *	–	10423.6	5401.6	2873.3	1568.4	908.5	–	2.0	3.9	7.3	13.3	22.9
LocusRoute	1796.0	919.9	474.1	249.5	147.0	97.1	1.0	2.0	3.8	7.2	12.2	18.5
Cholesky	2748.1	1567.3	910.5	545.8	407.7	398.1	1.0	1.8	3.0	5.0	6.7	6.9
Cholesky *	–	–	2282.2	1320.8	880.9	681.1	–	–	4.0	6.9	10.4	13.4
Water	12592.0	6370.8	3320.9	1705.5	897.5	451.3	1.0	2.0	3.8	7.4	14.0	27.9
Appbt	4928.3	2617.3	1360.5	704.7	389.7	223.7	1.0	1.9	3.6	7.0	12.6	22.0
Multigrid	2792.0	1415.6	709.1	406.2	252.9	165.5	1.0	2.0	3.9	6.9	11.0	16.9
CG	1279.2	724.9	498.0	311.1	179.0	124.9	1.0	1.8	2.6	4.1	7.1	10.2
EM3D	331.7	192.1	95.5	46.8	22.4	10.7	1.0	1.7	3.5	7.1	14.8	31.1
Gauss	1877.0	938.9	465.8	226.4	115.7	77.8	1.0	2.0	4.0	8.3	16.2	24.1
FFT	1731.8	928.0	491.8	261.6	136.7	71.8	1.0	1.9	3.5	6.6	12.7	24.1
SOR	1066.2	535.7	268.8	134.9	68.1	32.3	1.0	2.0	4.0	7.9	15.7	33.0
MICCG3D-32-Coarse	–	36.6	21.7	11.7	6.9	4.4	–	0.5	0.8	1.5	2.5	3.9
MICCG3D-32-Fine	–	–	11.7	5.8	2.9	1.5	–	–	1.5	3.0	5.9	11.5
MICCG3D-64-Coarse	–	–	–	–	–	32.2	–	–	–	–	–	4.3
MICCG3D-64-Fine	–	–	–	–	–	12.5	–	–	–	–	–	11.1

[Agarwal et. al. ISCA'95]

Alewife "Software" Directory

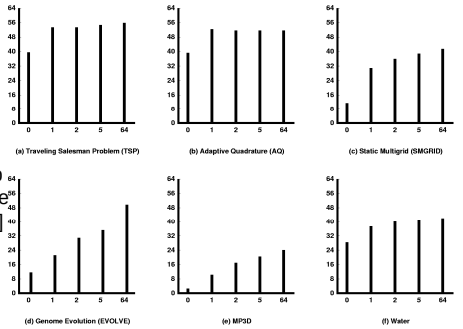
- Claim: Alewife performance only 2-3x worse with pure software directory management
- Only affects (slows) on memory side
  - still have cache mechanism on requesting processor side

Alewife Primitive Op Performance



[Chaiken+Agarwal, ISCA'94]

Alewife Software Data



[y: speedup
x: hardware pointers]

[Chaiken+Agarwal, ISCA'94]

Caveat

- We're looking at simplified version
- Additional care needed
  - write (non) atomicity
    - what if two things start a write at same time?
  - Avoid thrashing/livelock/deadlock
  - Network blocking?
  - …
- Real protocol states more involved
  - see HP, Chaiken, Culler and Singh...

Digesting…

Common Case Fast

- Common case
  - data local and in cache
  - satisfied like any cache hit
- Only go to messaging on miss
  - minority of accesses (few percent)

Model Benefits

- Contrast with completely software "Uniform Addressable Memory" in pure MP
  - must form/send message in all cases
- Here:
  - shared memory captured in model
  - allows hardware to support efficiently
  - minimize cost of "potential" parallelism
    - incl. "potential" sharing

General Alternative?

- This requires including the semantics of the operation deeply in the model
- Very specific hardware support
- Can we generalize?
- Provide more broadly useful mechanism?
- Allows software/system to decide?
  - (idea of Active Messages)

Big Ideas

- Simple Model
  - Preserve model
  - While optimizing implementation
- Exploit Locality
  - Reduce bandwidth and latency

Big Ideas

- Model
  - importance of strong model
  - capture semantic intent
  - provides opportunity to satisfy in various ways
- Common case
  - handle common case efficiently
  - locality

Big Ideas

- Hardware/Software tradeoff
  - perform common case fast in hardware
  - handoff uncommon case to software