ARCHITECTURE AND DESIGN OF ALPHASERVER GS320

GHARACHORLOO, ET. AL.

Presented By:
Ananya

Topics

- Motivation
- Physical Design Overview
- Directory Based Coherence Overview
- Coherence Protocol Features

Motivation

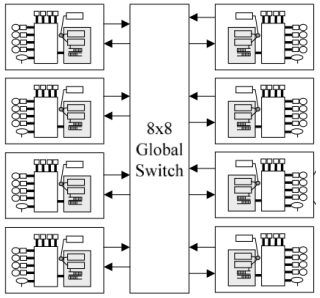
- Time frame: circa 2000
- Servers in high demands: 4-64 cores
  - Large scale (~100s nodes) multiprocessor seldom used
- Snoop-based protocols
  - Don't scale well beyond 4-8 cores
- Directory based protocols
  - Optimized for large scale multi-processors
  - Unnecessary overheads on limited scale systems
- Solution
  - Tailor the traditional directory based approach to mid-scale systems
    - Using simplified and specific hardware structure
    - Cutting corners on network transactions wherever possible (based on the h/w)

Topics

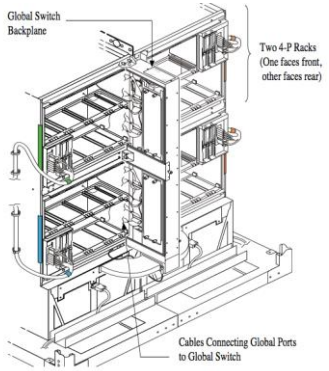
- Motivation
- Physical Design Overview
- Directory Based Coherence Overview
- Coherence Protocol Features

Physical Design Overview

- Hierarchical, shared memory multiprocessor
- Global Level
  - Global Switch
    - Centrally Buffered
    - Supports Virtual Lanes

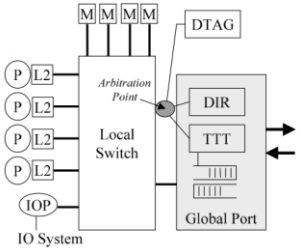


Physical Design Overview



Physical Design Overview

- Quad Building Block
  - 10 Port Local Switch
    - 4 Processors
    - 8X4 GB memory
    - 1 I/O Interface
      - Up to 8 PCI Buses
      - 64 Entry Cache
    - 1 Global Port
  - Coherence
    - Within a QBB
      - DTAG (duplicate tag store)
        - External copy of each processor's cache tags
    - Across QBB
      - Directory
        - 14 bit per 64 Byte line
      - Transactions-in-transit table (TTT)
        - Keeps track of pending transactions from a node
        - 48 entry, associative

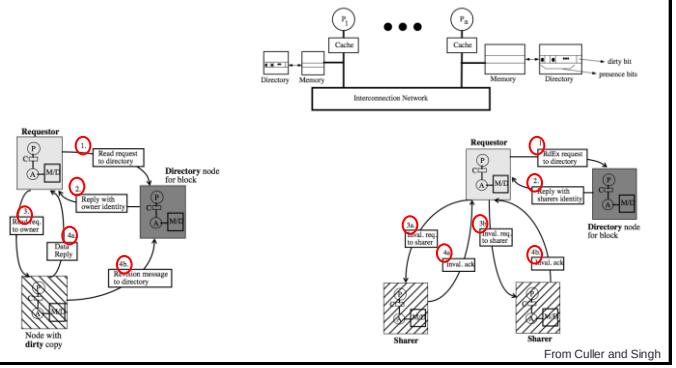


Topics

- Motivation
- Physical Design Overview
- Directory Based Coherence Overview
- Coherence Protocol Features

Directory Based Coherence Overview

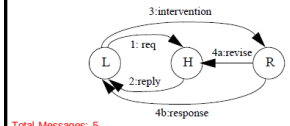
- Refreshing basic concepts



From Culler and Singh

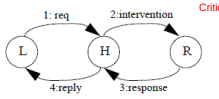
Directory Based Coherence Overview

- Common optimizations



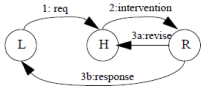
Total Messages: 5
Critical Messages:4

(a) Strict request-reply



Total Messages: 4
Critical Messages:4

(a) Intervention forwarding



Total Messages: 4
Critical Messages:3

(c) Reply forwarding

From Culler and Singh

Topics

- Motivation
- Physical Design Overview
- Directory Based Coherence Overview
- Coherence Protocol Features

Coherence Protocol

- 4 Types of Requests
  - Read
  - Read Exclusive
  - Exclusive
    - Have a shared copy, need to write to it
  - Exclusive w/o data
    - Needed if the entire cache line is going to be written to
    - No data required, just status upgrade

Coherence Protocol Optimizations

- Dirty sharing supported
  - Data can be shared without updating the home-node
- Elimination of
  - NACKs/Retries
  - Blocking at home node

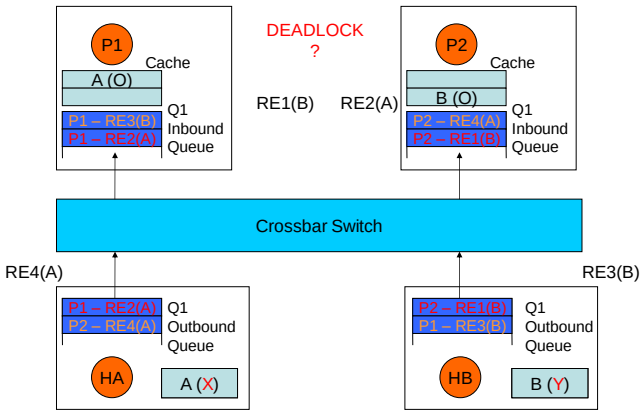
Coherence Protocol Optimizations

- All transactions complete with a single message to home
  - Owner node can always service a forwarded request
  - State changes occur immediately when home is first visited
  - No livelock/starvation/fairness problems
  - State machines at nodes can be simpler and faster
- Early commits can be utilized (discussed later)
- Concerns
  - Deadlocks?
  - Consistency models?

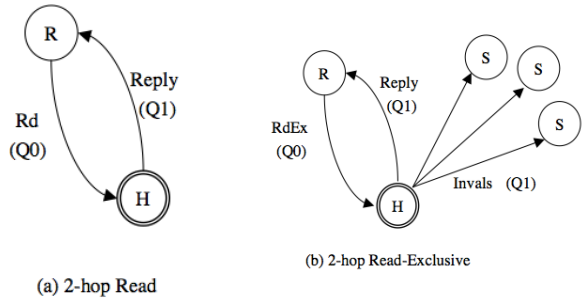
Deadlock Avoidance

- Three virtual lanes
  - Q0 : Requestor → Home
  - Q1 : Messages from Home
  - Q2 : Remote Node → Requestor
- Restrictions
  - Q1: Total Ordering
  - Q0: Point-to-point ordering

Quite Restrictive!
Really Necessary?



Protocol Transaction Flows

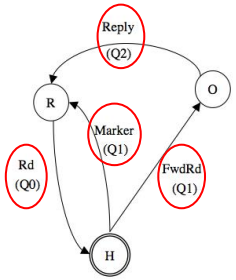


(a) 2-hop Read

(b) 2-hop Read-Exclusive

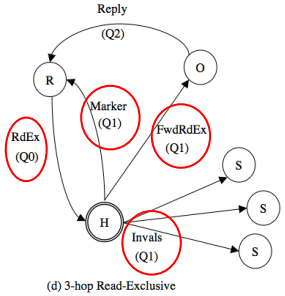
Protocol Transaction Flows

- Marker
  - Used for disambiguation
  - Conveys the order of operations to the same location, seen by home, to the requestor



(c) 3-hop Read

Protocol Transaction Flows

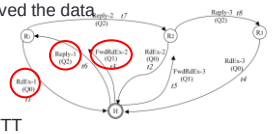


(d) 3-hop Read-Exclusive

Remember: Q1 is totally ordered!

Dealing With Request Races

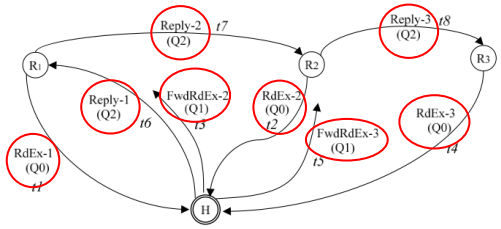
- Two races possible while forwarding a request
  - Late Request Race
    - Request arrives after owner has already written back the line
  - Early Request Race
    - Request arrives before owner has received the data



- Solutions
  - Late request
    - Keep to-be-eliminated locations in the TTT until the home node acknowledges write-back
  - Early request
    - Keep the request blocked in the incoming Q1 until the data is received

Example

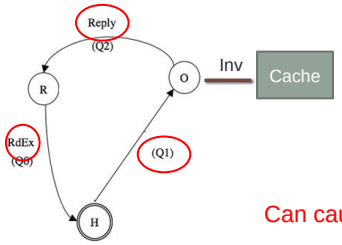
- Forwarding Chain:



More Optimizations and their effects on Consistency

Consistency

- Further optimizations
  - Using early acknowledgements (for shared, clean data invalidation)
  - Note: Instructions on the inbound queue can reorder



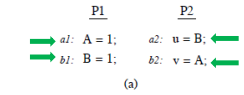
Can cause correctness issues!!

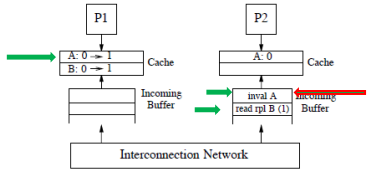
Consistency

- Early Acknowledgements
  - A case for inv to clean, shared line

Solutions:
- Disallow bypasses past invalidation reqs
- Service previously committed invs whenever program order is enforced

Assume SC:
{u,v}={1,0} Invalid



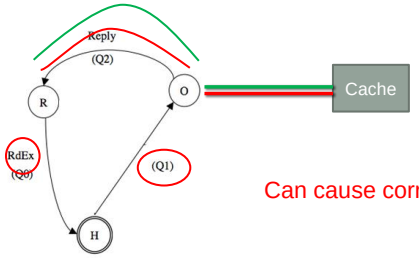


Still Queued!!

State: {u,v} = {1,0}
NOT ALLOWED!!

Consistency

- Early Acknowledgements
  - Extend to all other types of requests
  - Send commit first, and then the data
  - Allowed to go past the ordering points as long as commits for all requests have been received

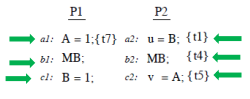


Can cause correctness issues!!

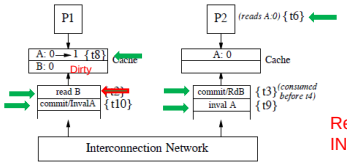
Consistency

- Early Acknowledgements for forwarding requests
  - Problem

{u,v}={1,0} Invalid



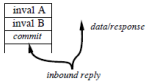
Solutions:
- Commit cannot bypass previous requests
- All requests to be serviced when program order enforced (membar)



Result - {u,v}={1,0} INCORRECT!!

Consistency

- Further optimizations
  - Problem: Long inbound path to processors
    - Memory ordering : mem barrier instruction (resulting in chunks of ops)
    - Arbitration at the interconnect (both local and global level)
  - Solution
    - Split incoming transactions
      - Commit: For ordering purposes
      - Data/Response: For using the inbound data early
    - Ensuring proper ordering?
      - Track the number of issued requests & commits received
      - Wait until both are equal whenever barrier is reached



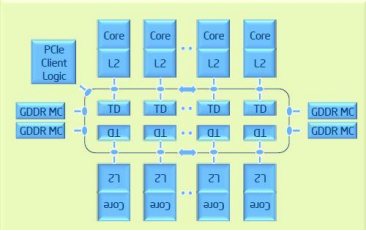
Summary

- Dirty Sharing
  - Reduces home node traffic
- No NACKs
  - Reduces network traffic
  - Allows simpler implementation
  - Removes livelocks, starvation, fairness problems
  - Can introduce deadlocks
    - Three dedicated queues
    - Total Q1 ordering
  - Can introduce two kinds of races
    - Late Request
      - Use TTT
    - Early Request
      - Block in the incoming Q1
- Early Commits & Split Responses
  - Compensates for queuing delays
  - Reduces waiting time

Need special rules for ensuring consistency

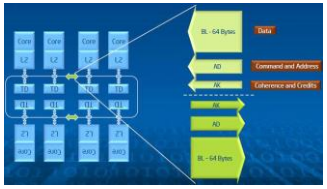
Annex: Intel Xeon Phi

- A modern multi-core system
  - 60 cores
  - Bi-directional ring interconnect



Intel Xeon Phi

- Each direction – 3 independent rings



- On an L2 miss:
  - Address request is sent on the address ring
  - If it is matched by a TD
    - Data is provides or a forward request is generated
  - If data is not found
    - Request is sent to the memory controller

Intel Xeon Phi

- Caches

Cache Hierarchy

Parameter	L1	L2
Coherence	MESI	MESI
Size	32 KB + 32 KB	512 KB
Associativity	8-way	8-way
Line Size	64 bytes	64 bytes
Banks	8	8
Access Time	1 cycle	11 cycles
Policy	pseudo LRU	pseudo LRU
Duty Cycle	1 per clock	1 per clock
Ports	Read or Write	Read or Write

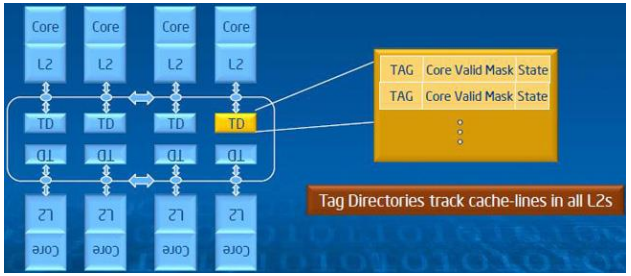
- Coherence

L2 Cache States

L2 Cache State	State Definition
M	Modified. Cacheline is updated relative to memory (GDDR). Only one core may have a given line in M-state at a time.
E	Exclusive. Cacheline is consistent with memory. Only one core may have a given line in E-state at a time.
S	Shared. Cacheline is shared and consistent with other cores, but may not be consistent with memory. Multiple cores may have a given line in S-state at the same time.
I	Invalid. Cacheline is not present in this core's L2 or L1.

- Dirty Sharing??

Intel Xeon Phi



Intel Xeon Phi

- Dirty Sharing
  - Tag Directory

Tag Directory States

Tag Directory State	State Definition
GOLS	Globally Owned, Locally Shared. Cacheline is present in one or more cores, but is not consistent with memory.
GS	Globally Shared. Cacheline is present in one or more cores and consistent with memory.
GE/GM	Globally Exclusive/Modified. Cacheline is owned by one and only one core and may or may not be consistent with memory. The Tag Directory does not know whether the core has actually modified the line.
GI	Globally Invalid. Cacheline is not present in any core.

References

- Architecture and Design of AlphaServer GS320
  - Gharachorloo, et. al.
- Xeon Phi System Software Guide
  - http://software.intel.com/sites/default/files/article/334766/intel-xeon-phi-systemsoftwaredevelopersguide.pdf
- Intel Xeon Phi Coprocessor (White Paper)
  - http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner
- Parallel Computer Architecture: A Hardware/Software Approach
  - Culler & Singh