AlphaServer GS320

Alexander Bowman and Andrew Hahn

1

Architecture Overview

- Supports up to 8 Quad-Processor Building Blocks (QBB)
- QBBs are connected via the Global Switch (GS)
- GS supports virtual lanes
- GS allows for totally ordered multicasts

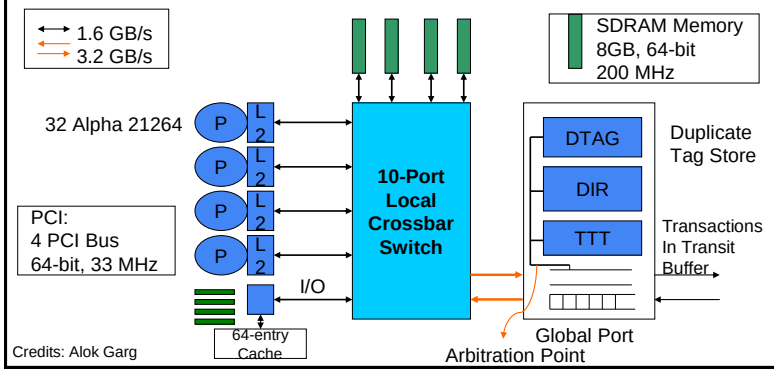
2

Quad-Processor Building Blocks (QBB)

- Supports up to 4 processors
- Uses duplicate tag store (DTAG)
- DTAG stores copy of each tag in each processor's cache
- DTAG handles intra-node coherence
- Directory (DIR) and transaction in transit table (TTT) handle inter-node coherence

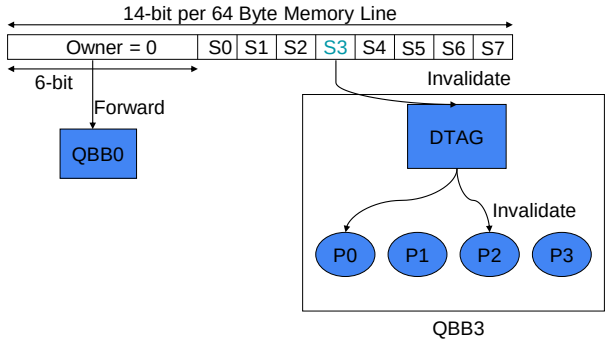
3

Quad-Processor Building Block (QBB)



4

The Directory



Credits: Alok Garg

5

Coherence Protocol Overview

- Exploits the limited number of processors
- Supports read, read-exclusive, exclusive, and exclusive without data
- No negative acknowledgements
- Dirty sharing - data can be shared without updating the home node
- Inter-node and intra-node coherence use mostly the same protocol

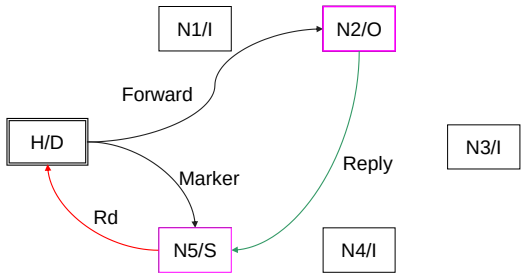
6

MOESI

- MESI + owner state
- Owner has write access but can share data with other processors
- On write owner invalidates sharers but does not bring home data up to date
- Home forwards requests to owner (if one exists)

7

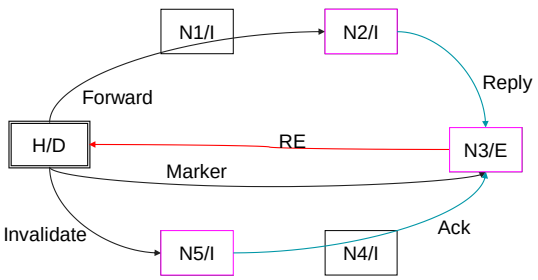
MOESI Read



Credits: Alok Gorg

8

MOESI Read-Exclusive



Credits: Alok Gorg

9

Avoiding Protocol Deadlock

- Example: Data is stuck behind forward request in queue
- Solution: Three virtual lanes Q0, Q1, Q2
- Q0: Requests from processor to home
- Q1: Home to processor (Totally Ordered!)
  - Total ordering: messages received in the same ordered they are sent
  - Ordering comes from the crossbar switch
- Q2: Third party processor

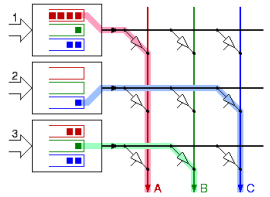
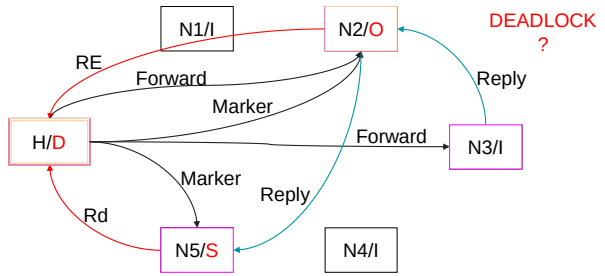


Image source: Alok Garg's slide deck

10

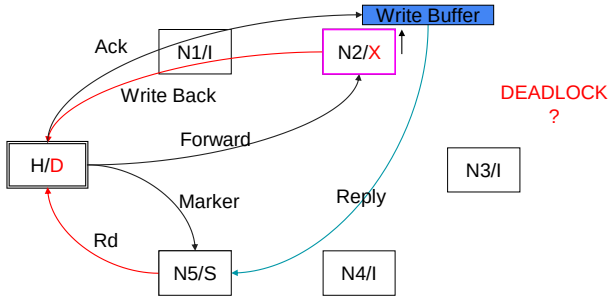
Early Request Race Condition



Credits: Alok Garg

11

Late Request Race Condition



Credits: Alok Garg

12

Efficient Implementation of Consistency Models

- AlphaServer GS320 implements two new optimizations
  - Separation of Incoming Read Replies
    - Commit
    - Data/Response
  - Early Commit for Read and Read-Exclusive Requests

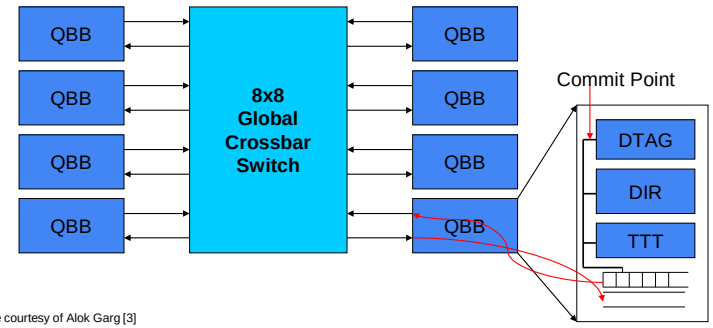
13

Early Acknowledgement of Invalidation Requests

- Invalidations are considered to be complete when a *commit event* is generated
- Commit incoming invalidation requests as soon as they arrive in the incoming queue
  - As opposed to when the invalidation has been completed
  - Don't have to wait for the invalidation to reach the head of the incoming queue in every processor with an old copy
  - Processors can still hold a stale value while the writing processor thinks all copies have been invalidated

14

Commit Points

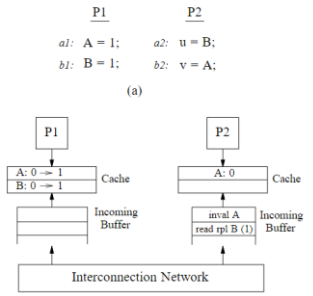


Slide courtesy of Alok Garg [3]

15

Early Acknowledgement: An Example

- (u,v) = (1,0) is disallowed in this example under sequential consistency, but is possible if early acknowledgements are applied without being careful
- With a careless implementation of early acknowledgment, (1,0) is possible



Source: Figure 4 from [1]

16

Maintaining Consistency with Early Acknowledgement

- Two solutions to the problem
  - Prevent read requests from bypassing invalidation requests
    - Total FIFO ordering isn't necessary
    - Well-suited to strict consistency models
  - Require previously committed invalidations to be serviced prior to a read to ensure program order
    - Effectively flush pending invalidations on a read reply
    - Works well for more relaxed consistency models where program order is enforced less often
- Is there anything we can do to avoid this overhead while ensuring consistency?

17

GS320's Memory Model

- GS320 uses the Alpha memory model
  - The programmer must explicitly set synchronization points for memory
  - Invalidations commit when they reach the arbitration point
    - Within the node for local requests
    - At the global switch for remote requests

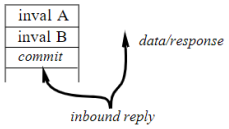
Relaxation:	W →R Order	W →W Order	R →RW Order	Read Others' Write Early	Read Own Write Early	Safety Net
IBM 370	✓					serialization instructions
TSO	✓				✓	RMW
PC	✓			✓	✓	RMW
PSO	✓	✓			✓	RMW, STBAR
WO	✓	✓	✓		✓	synchronization
RCsc	✓	✓	✓		✓	release, acquire, nsync, RMW
RCpc	✓	✓	✓	✓	✓	release, acquire, nsync, RMW
Alpha	✓	✓	✓		✓	MB, WMB
RMO	✓	✓	✓		✓	various MEMBARs
PowerPC	✓	✓	✓	✓	✓	SYNC

Source: Slide 162 of Professor Dwarkadas' lecture slides for this course [2]

18

Separation of Incoming Replies

- The choice to commit at the arbitration point results in long incoming queues
  - Flushing the queue on every barrier is expensive
  - Preventing read requests from bypassing invalidation requests delays critical data
- Separate requests into two components:
  - A reply with the requested data that can bypass the incoming queue altogether
  - A commit message used for ordering
- Replies can be sent on Q2 and commit messages can be sent on Q1



Source: Figure 5 from [1]

19

Separation of Incoming Replies

- Time critical data replies bypass other messages
- The Commit reply in the inbound queue keeps a FIFO ordering of incoming messages
  - Allows early invalidation acknowledgement to be applied without having to flush the queue on every read
    - We know what invalidations should have been performed prior to the read
- This process requires hardware support. Each processor must:
  - Expect two reply components
  - Have a count of pending requests to ensure all requests received prior to a barrier have been serviced before continuing execution

20

Early Commit for Read and Read-Exclusive Requests

- Applying early commit to data requests as well can reduce the delay at synchronization points
  - Processors can continue past barriers as long as they have received their commit replies (sent from the arbitration point), even if the data they require (sent from the processor) still has not arrived
  - From a consistency point of view, events are considered complete at commit time
- Commit events cannot bypass invalidates, reads, or read-exclusive data forwards to satisfy consistency

21

Generalized Applicability

- These optimizations can be used in any memory consistency model, regardless of how strict
- Reply Separation into commit and data messages is more useful for more relaxed models that see performance benefits from allowing reads to bypass other entries in the incoming queue
- Early commit drastically improves performance in strict consistency models as well as more relaxed models by allowing processors to pass synchronization points as soon as all commit messages have been received
- No rollback is ever required like in speculative models. Once an event is committed, it can be considered complete in terms of ordering

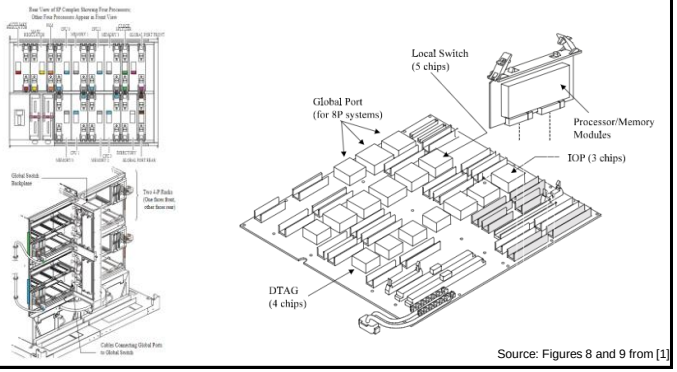
22

AlphaServer GS320 Hardware Implementation

- Implemented in 1999 with a technology stack from 1997-98
- Designed for modularity and ease of upgradability
- QBBs and I/O can be swapped out while the system is online
  - Streamlining the process of repairs and upgrades
- The modularity makes it possible to tailor the hardware to the specific application

23

AlphaServer GS320 Hardware Implementation



Source: Figures 8 and 9 from [1]

24

System Performance

- Less time spent spinning at barriers since it's not necessary to wait for data, only the commit message
- Lack of negative ACK messages reduces network traffic
- Deadlock is avoided through the total ordering on Q1
- Replies bypassing requests in Q1 (by going through Q2) improves read latency

Table 3: Effective latency for write operations.

Case	Pipelined Writes	Writes Separated by Memory Barriers
Local Home, No Sharers	58ns	387ns
Local Home, Remote Sharers	66ns	851ns
Remote Home, No Sharers	135ns	1192ns
Remote Home, Remote Sharers	148ns	1257ns

Table 5: Impact of separating the commit component and generating early commits.

Case	Back-to-Back Dependent Reads	Reads Separated by Memory Barriers
2-hop, Clean	960ns	1215ns
3-hop, Dirty	1478ns	1529ns

Source: Tables 3 and 5 from [1]

25

AlphaServer GS320's Influences in the Modern Day

Intel Rack Scale Design

- Resources can be dynamically assigned as required at runtime to suit the task at hand
  - 100G ethernet makes it possible to form virtual nodes that are composed from hardware in different locations on demand
- Finer control over system building blocks
  - FPGA, accelerator, and networking modules can be installed on sleds within a rack

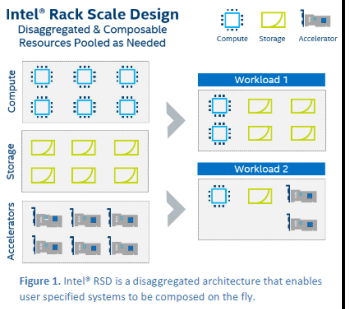


Figure 1. Intel® RSD is a disaggregated architecture that enables user specified systems to be composed on the fly.

Source: Intel [3]

26

Citations

[1] Gharachorloo, Kourosh, et al. “Architecture and Design of AlphaServer GS320.” ACM SIGARCH Computer Architecture News, vol. 28, no. 5, Dec. 2000, pp. 13–24, 10.1145/378995.378997. Accessed 28 Mar. 2022.

[2] Dwarkadas, Sandhya. “Shared Memory and Memory Consistency.” CSC2/458 Lecture Slides, 2022.

[3] Garg, Alok. Architecture & Design of AlphaServer GS320 Presentation, 2022. Animations used for examples.

[4] Intel Corporation. Intel Rack Scale Design Architecture. 2018.

27