Shared Memory Systems
Part III

Randal E. Bryant
CS 740
Dec. 3, 1997

Topics

• Fixing network-based coherency protocol
• Adding features
• Buffering & resource considerations
• Comparison to “real” protocols
  – SGI Origin
Network-Based Cache Coherency

Home-Based Protocol

- Each block has “home”
  - Memory controller tracking its status
  - Based on physical address
- Home maintains
  - Block status
  - Identity of copy holders
    - 1 bit flag / processor
  - Only need entry when block has remote copy

**Memory Controller 4**

<table>
<thead>
<tr>
<th>Block</th>
<th>Status</th>
<th>Copy Holders</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>shared</td>
<td>0101010101</td>
</tr>
<tr>
<td>25</td>
<td>Exclusive</td>
<td>0100000000</td>
</tr>
<tr>
<td>26</td>
<td>uncached</td>
<td>0000000000</td>
</tr>
</tbody>
</table>

**Block Status Values**

- **Shared**
  - 1 or more remote, read-only copies
- **Exclusive**
  - Single, writeable copy in remote cache
- **Uncached**
  - No remote copies
Implementation Details

- p  Processor identifier
- a  Block address
- d  Block data

Messages from Cache to Block Home

- Cache requests

<table>
<thead>
<tr>
<th>Type</th>
<th>Contents</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>p, a</td>
<td>Processor p wants to read a</td>
</tr>
<tr>
<td>XREAD</td>
<td>p, a</td>
<td>Processor p wants to write a</td>
</tr>
</tbody>
</table>

- Cache replies

<table>
<thead>
<tr>
<th>Type</th>
<th>Contents</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITEBACK</td>
<td>p, a, d</td>
<td>Proc. p flushes exclusive block</td>
</tr>
<tr>
<td>WRITEHOLD</td>
<td>p, a, d</td>
<td>Proc. p shares exclusive block</td>
</tr>
<tr>
<td>RELEASE</td>
<td>p, a</td>
<td>Proc. p flushes clean block</td>
</tr>
</tbody>
</table>

- Treat evictions as unsolicited replies
Implementation Details (Cont.)

- \( p \) Processor identifier
- \( a \) Block address
- \( d \) Block data

Messages from Block Home to Cache

- **Home requests**

<table>
<thead>
<tr>
<th>Type</th>
<th>Contents</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALIDATE</td>
<td>( p, a )</td>
<td>Give up clean copy</td>
</tr>
<tr>
<td>FETCH</td>
<td>( p, a )</td>
<td>Get readable copy from remote</td>
</tr>
<tr>
<td>XFETCH</td>
<td>( p, a )</td>
<td>Get writeable copy from remote</td>
</tr>
</tbody>
</table>

- **Home replies**

<table>
<thead>
<tr>
<th>Type</th>
<th>Contents</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>( p, a, d )</td>
<td>Reply to read / write request</td>
</tr>
<tr>
<td>NACK</td>
<td>( p, a )</td>
<td>Cannot fulfill request</td>
</tr>
</tbody>
</table>
Tricky Issues

Independent Actions between Cache & Home

• Cache evicts exclusive block while home in process of fetching
  – Home will get WRITEBACK when expecting WRITEHOLD
  – Cache will receive FETCH for invalid block
• Cache evicts exclusive block while home in process of xfetching
  – Home will receive WRITEBACK generated by cache
  – Cache will receive XFETCH for invalid block

Independent Actions between Two Processor Caches

• Processor r attempts read or write while home handling read or write request by processor p
  – Home sends NACK to processor r
  – Processor r retries read or write operation
• Potential for livelock
  – Processor r keeps getting NACK’ed
Performing Processor Operations

- Processor requests cache to perform load or store
  - On word in cache block i
- Cache line currently holds block t
  - May or may not have i = t
- Cache can either:
  - Perform operation using local copy
  - Issue request message to network (*italicized*)
    » Wait for reply (*bold*)
    » Stall until completed
Cache Handling of Processor Read

- **Invalid**
  - Read Miss \( \text{READ}(p,i) \)
  - Read Miss \( \text{WRITEBACK}(p,t,d) \)

- **Shared**
  - Read Miss \( \text{RELEASE}(p,t) \)
  - DATA \( (p,i,d) \)

- **Exclusive**
  - Read Miss \( \text{NACK}(p,i) \)

- **Read Pending**
  - Read Miss \( \text{READ}(p,i) \)

**Transitions**:
- Read Hit
- Writeback
- Release
Processing by Home

Home Memory Controller

- Receives requests from caches
  - To handle processor reads & writes
  - Unsolicited evictions
- May need to retrieve or invalidate remote copies
  - Sends requests to copy holders
  - Copy holders send replies
- Replies to requestor
- Send NACK if unable to satisfy request
Home Handling of Processor Read

- **READ** \((r,i)\)
- **NACK** \((r,i)\)
- **XREAD** \((r,i)\)
- **NACK** \((r,i)\)
- **READ** \((p,i)\)
- **FETCH** \((q,i)\)
- **WRITEHOLD** \((q,i,d)\)
- **DATA** \((p,i,d)\)
- **WRITEBACK** \((q,i,d)\)
- **DATA** \((p,i,d)\)
- **READ** \((p,i)\)
- **DATA** \((p,i,d)\)
- **WRITEBACK** \((q,i,d)\)
- **DATA** \((p,i,d)\)
- **UNSOLICITED EVICTION**

**Definitions**
- **S** Set of copy holders
  - `+=` Add to set
  - `-=` Remove from set

**Terms**
- **p** Requesting Processor
- **q** Copy-holding Processor
- **r** Competing Processor(s)
Cache Handling of Processor Write

- Invalid
- Shared
- Write Miss $\text{RELEASE}(p,t)$
- NACK$(p,i)$
- Write Miss $\text{XREAD}(p,i)$
- Write Pending
- DATA$(p,i,d)$
- Exclusive
- Write Hit

Write Miss $\text{WRITEBACK}(p,t,d)$

Requested Block:
i
Current Block:
t
Home Handling of Processor Write

- Uncached
  - \(XREAD(p, i)\)
  - \(DATA(p, i, d)\)
  - \(S + = p\)

- Exclusive
  - \(XREAD(p, i)\)
  - \(XFETCH(q, i)\)
  - \(WRITEBACK(q, i, d)\)
  - \(DATA(p, i, d)\)
  - \(S + = p\)
  - \(S - = q\)
  - \(S\) empty

- Dirty Write Pending
  - \(READ(r, i)\)
  - \(NACK(r, i)\)
  - \(XREAD(r, i)\)
  - \(NACK(r, i)\)

- Clean Write Pending
  - \(RELEASE(q, i)\)
  - \(S - = q\)
  - \(S\) nonempty

- Shared
  - \(XREAD(p, i)\)
  - \(Forall\ q\ in\ S:\ INVALIDATE(q, i)\)

- S Set of copy holders
  - += Add to set
  - -= Remove from set

- p Requesting Processor
- q Copy-holding Processor(s)
- r Competing Processor(s)
Network Monitoring by Cache

- **Cache receives commands from network**
  - Unlike snooping, only get messages regarding currently-held blocks

- **Possible actions**
  - Invalidate entry and release copy
  - Allow sharing of exclusively-held block
  - Surrender copy of block
Cache Handling of Network Commands

Invalidate \((p,t)\)

Invalid

Blocks that have already been written back or released

Invalidate \((p,t)\)

Release \((p,t)\)

Shared

Exclusive

Invalidate \((p,t)\)

Fetch \((p,t)\)

Writehold \((p,t,d)\)

XFetch \((p,t)\)

Writeback \((p,t,d)\)

Current Block

t
Home Handling of Unsolicited Evictions

- **UNCACHED**
  - \( S \text{ empty} \)
  - \( WRITEBACK(q, i, d) \)
  - \( S \leftarrow q \)

- **EXCLUSIVE**
  - \( S \text{ empty} \)

- **SHARED**
  - \( S \text{ nonempty} \)
  - \( RELEASE(q, i) \)
  - \( S \leftarrow q \)

\( q \) Copy-Holding Processor

\( S \) Set of copy holders
- \( +\) Add to set
- \( -\) Remove from set
Optimization 1: Make Clean Copy Exclusive

Cache Request

\texttt{GIVEME \ p, a} \quad \text{Want exclusive version of currently-held block}

Home Response

\texttt{ACK \ p, a} \quad \text{Permission granted}
\texttt{NACK \ p, a} \quad \text{Cannot fulfill request}

Motivation

• Avoids need to transmit data
  – Minor consideration
• Potential for fast acknowledgement
  – Next optimization
• Allows processor to detect whether any writes since last read
  – If receive \texttt{ACK}
  – To implement store-conditional
Cache Handling of Processor Write

Invalid

Write Miss
RELEASE(p,t)

NACK(p,i)

Write Miss
XREAD(p,i)

Write Pending

Shared

Write Hit
GIVEME(p,i)

Giveme Pending

NACK(p,i)

DATA(p,i,d)

Exclusive

ACK(p,i)

Write Hit

WRITEBACK(p,t,d)

Requested Block

Current Block

i

t
Home Handling of Processor Write

- **S**: Set of copy holders
- **+=**: Add to set
- **-=**: Remove from set

**States and Transitions**

- **Clean**
  - **Data**: \( \text{DATA}(p,i,d) \)
  - **Release**: \( \text{RELEASE}(q,i) \)
  - **Invalidate**: \( \text{INVALIDATE}(q,i) \)
  - **Release**: \( \text{RELEASE}(q,i) \)
  - **Set empty**: \(|S| = 1\)
  - **ACK**: \( \text{ACK}(p,i) \)

- **Shared**
  - **GIVEME**: \( \text{GIVEME}(p,i) \)
  - **NACK**: \( \text{NACK}(r,i) \)
  - **GiveMe**: \( \text{GIVEME}(r,i) \)
  - **Release**: \( \text{RELEASE}(q,i) \)
  - **Set nonempty**: \(|S| > 1\)
  - **NACK**: \( \text{NACK}(r,i) \)
  - **GiveMe**: \( \text{GIVEME}(r,i) \)
  - **Release**: \( \text{RELEASE}(q,i) \)
  - **Set empty**: \(|S| = 1\)
  - **ACK**: \( \text{ACK}(p,i) \)

- **Exclusive**
  - **Copy-holding**: \( \text{XREAD}(r,i) \)
  - **NACK**: \( \text{NACK}(r,i) \)
  - **GiveMe**: \( \text{GIVEME}(r,i) \)
  - **Release**: \( \text{RELEASE}(q,i) \)
  - **Set nonempty**: \(|S| > 1\)
  - **NACK**: \( \text{NACK}(r,i) \)
  - **GiveMe**: \( \text{GIVEME}(r,i) \)

**Variables**

- **p**: Requesting Processor
- **q**: Copy-holding Processor(s)
- **r**: Competing Processor(s)
Optimization 2: Fast Acknowledgement

Idea
- Enable processor write before other copies invalidated
- Only works if home has valid copy

Motivation
- Reduces write latency

Risk
- Possible sequential inconsistency
- Other processors continue to read old data
Home Handling of Processor Write

Uncached

- \text{XREAD}(p, i)
- \text{DATA}(p, i, d)

Exclusive

- \text{XREAD}(p, i)
- \text{XFETCH}(q, i)

Dirty Write Pending

- \text{READ}(r, i)
- \text{NACK}(r, i)
- \text{XREAD}(r, i)
- \text{NACK}(r, i)

Clean Write Pending

- \text{XREAD}(p, i)
- \text{DATA}(p, i, d)

S += p

\text{RELEASE}(q, i)
S -= q

\text{WRITEBACK}(q, i, d)
S += p

\text{DATA}(p, i, d)
S -= q

|S| > 1

|S| = 1

For all q in S-\{p\}:

- \text{INVALIDATE}(q, i)
- \text{RELEASE}(q, i)
- S -= q

\text{XREAD}(p, i)
\text{GIVEME}(p, i)
\text{ACK}(p, i)
\text{GIVEME}(r, i)
\text{NACK}(r, i)

\text{NACK}(r, i)
\text{NACK}(r, i)
\text{NACK}(r, i)

\text{NACK}(r, i)
\text{NACK}(r, i)
\text{NACK}(r, i)

p Requesting Processor
q Copy-holding Processor(s)
r Competing Processor(s)
S Set of copy holders
+= Add to set
-= Remove from set
Optimization 3:
Data Forwarding

Idea
- Copy holding processor sends data directly to requestor
- Also sends message to home

Motivation
- Reduce latency

Challenges
- New possibilities for deadlock & inconsistency
- Protocol extension left as exercise
Deadlock Principles

Conditions for Deadlock

- **Finite resources**
  - Network links
  - Buffers
- **Circular dependence**
  - Independent agents A, B, ...  
  - Require same resources
  - Acquire in different order

Avoidance

- Provide independent resources
- Acquire in same order
- Preemption
  - Force agent to give up already-acquired resource

A
Acquire X

B
Acquire Y

Acquire Y

Deadlocked

Acquire X
Resources for Memory Transaction

Sources of Resource Contention

- Processor / memory system
  - Resolve by stalling processor
  - Processor cannot hold onto memory resources
- Handling single memory transaction
  - Real possibility
- Handling multiple memory transactions by single processor
  - Generally serialize if cannot guarantee success
- Handling multiple memory transactions by independent processors
  - Real possibility
Protocol Deadlock Avoidance

Request-Reply Dependencies

• Provide separate routing resources for request & reply messages
  – Physically distinct or separate virtual channels

• Require buffer for reply to be allocated before make request
  – Stall processor if buffers not available at cache

Contentions Eliminated

• Between any request & any reply

• Between any two replies
  – Resources allocated at receiver beforehand
  – Processing of reply at receiver serves only to release resources
Out of Order Message Delivery

Assumed FIFO Ordering Between Sender & Receiver

- Cannot guarantee between separate request & reply networks
  - Attempting to reorder at receiver could lead to deadlock

Example Problem

- Cache sends \textit{READ} request
- Home sends DATA reply
  - Reply network
- Home sends INVALIDATE request due to some other transaction
  - Request network
- Messages received at cache out-of-order
  - Get INVALIDATE while in Pending Read state

Handling

- Respond with \textit{RELEASE}
- Wait for & discard DATA
- Retry

Other Problems Still Exist!
Protocol Deadlock Avoidance (Cont.)

Request-Request Dependencies

- Request by cache can generate request(s) by home
  - May not have sufficient buffer resources
  - May cause network contention
- Home can reply with NACK if cannot guarantee resource availability
  - Processor will stall while cache retries
- Timeout
  - Home can give up on pending operation
  - Release resources
  - Respond to requestor with NACK
Fairness / Livelock

Livelock Possibilities
- Cache will keep getting NACK’ed
- Processor will be unable to complete read or write operation

Is This a Problem?
- Don’t know of protocol that guarantees fairness
- Even if single memory operation guaranteed to complete, hard to guarantee fairness of synchronization protocol

Avoidance
- Each memory transaction has priority
- Dynamically increase as get NACK’ed
- Preempt lower priority transactions
SGI Origin System

Similar to Example Protocol

• Both derivatives of Stanford DASH protocol

Enhanced Implementation Features

• Designed to handle out-of-order messages at all levels
  – Important adaptive routing

• Support “clean exclusive” state
  – Give to read request when block unshared
  – Expedites subsequent write

• Forwarding of data from remote copy holder
  – Also in DASH

• Evictions of clean blocks don’t generate network traffic

• More effective handling of request-request deadlock avoidance
  – Send back information needed for requesting cache to handle invalidations

• Variable granularity directory representation
  – 64-bit vector represents either 64 clustered processors or 64 processor sets
SGI Origin Performance

Processor

• 195 MHz R10000

Latencies

<table>
<thead>
<tr>
<th>Level</th>
<th>Nanoseconds</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>L2 cache</td>
<td>56</td>
<td>11</td>
</tr>
<tr>
<td>Local memory</td>
<td>310</td>
<td>60</td>
</tr>
<tr>
<td>4P avg. remote memory</td>
<td>540</td>
<td>105</td>
</tr>
<tr>
<td>128P avg. remote memory</td>
<td>945</td>
<td>184</td>
</tr>
</tbody>
</table>

Dealing with High Latency

• Support prefetch operations
• Automatic page migration in virtual memory system