Shared Memory Systems

Randal E. Bryant
CS 740
Nov. 19, 1997

Topics

• Consistency Models
• Single Bus Systems
  – Coherence based on snooping
• Synchronization Programs
• Network-Based Systems
  – Coherence based on directories
Shared Memory Model

Conceptual View

- All processors access single memory
  - Physical address space
  - Use virtual address mapping to partition among processes
- If one processor updates location, then all will see it
  - Memory consistency
Bus-Based Realization

Memory Bus
- Handles all accesses to shared memory

Caches
- One per processor
- Allows local copies of heavily used data
- Must avoid stale data

Considerations
- Small step up from single processor system
  - Support added to many microprocessor chips
- Does not scale well
  - Bus becomes bottleneck
  - Limited to ~16 processors
Network-Based Realization

Memory
- Partitioned Among Processors

Network
- Transmit messages to perform accesses to remote memories

Caches
- Local copies of heavily used data
- Must avoid stale data
  - Harder than with bus-based system
  - Lots of things happening simultaneously

Considerations
- Scales well
  - 1024 processor systems have been built
- Nonuniform memory access
  - 100’s of cycles for remote access
Memory Consistency

Model

- Independent processes with access to shared variables
- No assumptions about relative timing of processes
  - Which starts first
  - Which runs fastest

Sequential Consistency

- Each process executes its steps in program order
- Overall effect should match that of some interleaving of the individual process steps

Initially: \( x = y = 0 \)

<table>
<thead>
<tr>
<th>Process A</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1: ( x = 1 )</td>
</tr>
<tr>
<td>a2: if (( y == 0 )) ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process B</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1: ( y = 1 )</td>
</tr>
<tr>
<td>b2: if (( x == 0 )) ...</td>
</tr>
</tbody>
</table>
Sequential Consistency Example

Process A

a1: x = 1
a2: if (y == 0) ...

Process B

b1: y = 1
b2: if (x == 0) ...

Possible Interleavings

<table>
<thead>
<tr>
<th>a1</th>
<th>a2</th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Sequential Inconsistency

• Cannot have both tests yield T
  – b2 must precede a1
  – a2 must precede b1
  – Cannot satisfy these plus program order constraints

Real Life Scenario

• Process A
  – Remote write x
  – Local read y
• Process B
  – Remote write y
  – Local read x
• Could have both reads yield 0
Snoopy Bus-Based Consistency

Caches
- Write-back
  - Minimize bus traffic
- Monitor bus transactions when not master

Cached blocks
- Clean block can have multiple, read-only copies
- To write, must obtain exclusive copy
  - Marked as dirty

Getting copy
- Make bus request
- Memory replies if block clean
- Owning cache replies if dirty
Implementation Details

Block Status

- Maintained by each cache for each of its blocks
- Invalid
  - Entry not valid
- Clean
  - Valid, read-only copy
  - Matches copy in main memory
- Dirty
  - Exclusive, writeable copy
  - Must write back to evict

Bus Operations

- Read
  - Get read-only copy
- Invalidate
  - Invalidate all other copies
  - Make local copy writeable
- Write
  - Write back dirty block
  - To make room for different block
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 bus request to get block
    » Stall processor until block ready

<table>
<thead>
<tr>
<th>P/B</th>
<th>Request Operation</th>
<th>i:t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bus Operation</td>
<td>Bus Block</td>
</tr>
<tr>
<td></td>
<td>Tag Update</td>
<td>Processor Operation</td>
</tr>
</tbody>
</table>

Action Key

<table>
<thead>
<tr>
<th>Block = i</th>
<th>Status</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>clean</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>Read / Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Done / Stall</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Processor

Cache

Bus Op

Bus Block

Bus Data
Bus Master Actions

- Invalid:
  - P Read ≠ Write
  - P Write ≠ Write
  - P Write ≠ Inval.
  - P Read = None

- Clean:
  - P Read = None
  - P Write = None

- Dirty:
  - P Read ≠ Write
  - P Write ≠ Write
  - P Write ≠ Inval.
  - P Read = None

- Requested Block
- Current Block

P/B Request Operation
i:t

Bus Operation
Tag Update
Processor Operation

Bus Block

- Write
- Read
- Stall
Bus Monitoring

- Cache monitors bus traffic when not master
  - Looks for operations on blocks matching cache entries
- Possible actions
  - Invalidate entry
  - Allow sharing of exclusively held block
    » Supply data on bus

Action Key

<table>
<thead>
<tr>
<th>P/B</th>
<th>Bus Operation</th>
<th>i:t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>—</td>
<td>Cache Operation</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- Processor
- Cache: Status, Tag, Data
  - clean, t
- Bus Op
- Bus Block = i
- Bus Data
Bus Snoop Actions

Invalid

Clean

Dirty

i Requested Block
t Current Block

Data: Cache supplies block

P/B Bus Operation i:t

B Inv =

B Read =

B ≠

Data

Cache Operation

CS 740 F’97
Example 1

Process A

\[ a1: \ x = 1 \]
\[ a2: \text{if} \ (y == 0) \ldots \]

Process B

\[ b1: \ y = 1 \]
\[ b2: \text{if} \ (x == 0) \ldots \]

Bus Transactions

\begin{align*}
\text{A: Read } x \\
\text{A: Invalidate } x \\
\text{A: Read } y \\
\text{A: Read } y \\
\text{B: Invalidate } y \\
\text{B: Read } x
\end{align*}
Example 2

**Process A**

- **a1:** \( x = 1 \)
- **a2:** if \( y == 0 \) ...

**Process B**

- **b1:** \( y = 1 \)
- **b2:** if \( x == 0 \) ...

**Bus Transactions**

**A**

- **a1:** \( x = 1 \)
- **a2:** = \( F \)

**B**

- **b1:** \( y = 1 \)
- **b2:** = \( F \)
Livelock Example

Process A

\[ a_1: \quad y = 0 \]

Process B

\[ b_1: \quad \text{while } ((t=y) \neq 0) \]
\[ b_2: \quad y = t+1 \]

Bus Transactions

A

- A: Read y
- B: Read y
- B: Invalidate y

Never gets chance to write

B

- A: Read y
- B: Read y
- B: Invalidate y

- A: Read y
- B: Read y

CS 740 F'97
Livellock Solution

New Bus Operation

• XRead
  – Combines Read + Invalidate as atomic transaction
    » Get copy of data
    » All other copies invalid

```
  P  Write  –
  Read   i
   i  Stall
```

```
  P  Write  ≠
  Read   i
   i  Stall
```

```
  P  Write  ≠
  XRead  i
   i  Write
```

```
  P  Write  =
  Inval.  i
   –  Write
```

```
  P/B  Request Operation  i:t
  Bus Operation  Bus Block
  Tag Update  Processor Operation
```

```
i  Requested Block

i  Current Block
```
Liveloop Solution (Cont.)

- Invalid
- Dirty
- Clean

B \times \text{Rd} = 
- 
- 
- 

\text{Data: Cache supplies block}

\begin{array}{c|c|c}
\hline
\text{B} & \text{XRd} & = \\
\text{--} & \text{--} & \\
\text{--} & \text{--} & \\
\text{--} & \text{Data} & \\
\hline
\end{array}

\text{i: Requested Block} \\
\text{t: Current Block} \\
\text{Data: Cache supplies block}

\begin{array}{c|c|c}
\hline
\text{P/B} & \text{Bus Operation} & \text{i:t} \\
\text{--} & \text{--} & \\
\text{--} & \text{Cache Operation} & \\
\hline
\end{array}
Prevented Liveloock Example

Process A

a1: y = 0

Process B

b1: while ((t=y) != 0)
b2: y = t+1

Bus Transactions

A

B: Read y
B: Invalidate y
A: XRead y
B: Read y

B

b1: t = y
b2: y = t+1

Fair bus cannot exclude request indefinitely
Single Bus Machine Example

SGI Challenge Series
- Up to 36 MIPS R4400 processors
- Up to 16 GB main memory

Bus
- 256-bit wide data
- 40-bit wide address
- Data transferred at 1.22 GB / second
- Split transaction
  - Read request & Read response are separate bus transactions
  - Can use bus for other things while read outstanding
  - Complicates synchronization

Performance
- 164 processor cycles to handle remote read
- Assuming no bus contention
Program Synchronization

Typical Scenario

- Processes need access to shared resources
  - Tables, I/O devices, program state
- Use software locks to prevent simultaneous access
  - Shared variables + synchronization conventions

Naive Synchronization

Obtain #1

```
while (lock != 0) {
    ;
    lock = 1;
}
```

Release

```
lock = 0;
```
Synchronization with Test-And-Set

Test-And-Set

- Atomically test value of variable and sets it to 1
- Could implement with cache protocol in manner similar to write
  – Even write needs to read in block since only overwrites one word

Modified Synchronization Code

```c
TST(x):
    temp = x;
    x = 1;
    return temp;
```

```c
while (TST(lock))
    ;
```
Scenario with Obtain #2

- Everyone fighting for exclusive copy of block
- Lots of traffic with no useful work
Revised Test-And-Set Synchronization

Modified Synchronization Code

Obtain #3

```
  do {
    while (lock)
    ;
  } while (TST(lock));
```

- Can used shared (read-only) copy of variable
Scenario with Obtain #3

Critical Section

lock = 0

TST(lock)

Bus Transactions

P1: Read lock
P2: Read lock
P3: Read lock

(Spinning on local copies)

P0: XRead lock
P2: Read lock
P3: Read lock
P3: XRead lock
P1: XRead lock
Other Synchronization Primitives

Atomic Exchange

\[ \text{exch } Rd, D(Rb) \]
- Atomically swap register \( Rd \) and memory location \( Rb+D \)
- Implement TST(lock) as:
  \[
  \begin{align*}
  &\text{li } r2, 1 \\
  &\text{exch } r2, \text{lock}
  \end{align*}
  \]

Load-Linked / Store-Conditional

- Used in MIPS
  \[
  \begin{align*}
  &\text{ll } Rt, D(Rb) \\
  &\text{sc } Rt, D(Rb)
  \end{align*}
  \]
- Similar to load
  - Set internal “link register” to effective (physical) address
- Similar to store
  - Abort if effective address matches link register, but this address modified since most recent load-linked
  - Indicate success or failure by setting \( Rt \) to 1 or 0

Obtain #4

\[
\begin{align*}
\text{loop: } &\text{ll } r3, \text{lock} \\
&\text{bne } r3, 0, \text{loop} \\
&\text{li } r3, 1 \\
&\text{sc } r3, \text{lock} \\
&\text{beq } r3, 0, \text{loop}
\end{align*}
\]
Network-Based Cache Coherency

Home-Based Protocol

- Each block has “home”
  - Memory controller tracking its status
- Home maintains
  - Block status
  - Identity of copy holders
    » 1 bit flag / processor

Block Status Values

- Shared
  - 1 or more remote, read-only copies
- Remote
  - Writeable copy in remote cache
- Uncached
  - No remote copies
Network-Based Consistency

To Obtain Copy of Block

- Processor sends message to its home
- Home retrieves remote copy if status is remote
- Sends copy to requester
- If exclusive copy requested, send invalidate message to all other copy holders

Tricky Details

- Lots of possible sources of deadlock & errors
- Don’t have serialization of events imposed by bus
- Transactions only “seen” by sender & receiver