TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems

Wookeun Jung

Multicore Computing Research Laboratory
Seoul National University
Backgrounds

• Distributed Shared Memory (DSM)
  • Sharing Data between Processors that Do Not Share Physical Memory
    • Give an image of shared memory
  • Easier to Program, Compared to using Message Passing Interface (MPI)
Backgrounds

• Previous DSM systems have two problems

• Lack of Portability
  • Hardware DSM
    • Supported for a specific device
  • Software DSM
    • Need kernel modification

• Lack of Scalability
  • Communication Overheads
Backgrounds

• Previous DSM systems have two problems

  • Lack of Portability -> Portable Software Platform
    • Hardware DSM
      • Supported for a specific device
    • Software DSM
      • Need kernel modification

  • Lack of Scalability
    • Communication Overheads -> Lazy Release Consistency
Backgrounds

• Previous DSM systems have two problems

• Lack of Portability -> **Portable Software Platform**
  • Hardware DSM
    • Supported for a specific device
  • Software DSM
    • Need kernel modification

• Lack of Scalability
  • Communication Overheads -> **Lazy Release Consistency**
  • False Sharing
Backgrounds

• Previous DSM systems have two problems

• Lack of Portability -> **Portable Software Platform**
  • Hardware DSM
    • Supported for a specific device
  • Software DSM
    • Need kernel modification

• Lack of Scalability
  • Communication Overheads -> **Lazy Release Consistency**
  • False Sharing -> **Multiple Writer Protocol**
Outline of This Presentation

• Backgrounds

• Lazy Release Consistency
  • Release Consistency
  • ERC vs. LRC
  • Interval, Timestamp, and Write notice

• Multiple-writer protocol
  • False sharing
  • Twin & Diff
  • Lazy Diff Creation

• Implementation
  • Data structures
  • Other details

• Evaluation
  • Scalability
  • ERC vs. LRC
Release Consistency

- Acquire and Release is used
- All the modifications done by a processor that **Releases** a Lock should be applied to a processor that **Acquires** a Lock
Release Consistency

• Acquire and Release is used
• All the modifications done by a processor that Releases a Lock should be applied to a processor that Acquires a Lock
Release Consistency

• Acquire and Release is used

• All the modifications done by a processor that Releases a Lock should be applied to a processor that Acquires a Lock
Release Consistency

• Acquire and Release is used
• All the modifications done by a processor that **Releases** a Lock should be applied to a processor that **Acquires** a Lock
Release Consistency

• Acquire and Release is used
• All the modifications done by a processor that Releases a Lock should be applied to a processor that Acquires a Lock
Release Consistency

• Acquire and Release is used
• All the modifications done by a processor that Releases a Lock should be applied to a processor that Acquires a Lock
Release Consistency

- Acquire and Release is used
- All the modifications done by a processor that **Releases** a Lock should be applied to a processor that **Acquires** a Lock
Release Consistency

- Acquire and Release is used
- All the modifications done by a processor that **Releases** a Lock should be applied to a processor that **Acquires** a Lock

Processor 1

- Acq(L)
- W(a)1
- Rel(L)

Processor 2

- W(a)0
- Acq(L)
- R(a)1

Can’t we be lazier?
Release Consistency

- Acquire and Release is used
- All the modifications done by a processor that **Releases** a Lock should be applied to a processor that **Acquires** a Lock
Release Consistency

- Acquire and Release is used
- All the modifications done by a processor that **Releases** a Lock should be applied to a processor that **Acquires** a Lock

![Lazy Release Consistency Diagram]

Diagram showing the sequence of events in processors 1 and 2 with arrows indicating the order and timing of acquire (Acq), release (Rel), and write operations (W).
ERC vs. LRC

Processor 1

Acq(L)  W(a)1  Rel(L)

Processor 2

Acq(L)  R(a)1  R(b)2

Processor 3

Acq(L)  W(b)2  Rel(L)
ERC vs. LRC

Processor 1

Acq(L) → W(a)1 → Rel(L)

Assumption: Variable a and b is not in the same page.

Processor 2

Acq(L) → R(a)1 → R(b)2

Processor 3

Acq(L) → W(b)2 → Rel(L)
ERC vs. LRC

Processor 1

Acq(L) → W(a)1 → Rel(L)

Processor 2

Acq(L) → R(a)1 → Rel(L)

Processor 3

Acq(L) → W(b)2 → Rel(L)

 ERC

\[ a = 1 \]

\[ b = 2 \]
ERC vs. LRC

Processor 1

Acq(L)  W(a)1  Rel(L)

Processor 2

Acq(L)  R(a)1  R(b)2

a = 1

Processor 3

Acq(L)  W(b)2  Rel(L)

b = 2

Unnecessary communication
ERC vs. LRC

Processor 1

Acq(L) \quad W(a)1 \quad Rel(L)

Processor 2

Acq(L) \quad R(a)1 \quad R(b)2

Processor 3

Acq(L) \quad W(b)2 \quad Rel(L)

LRC

a = 1

b = 2
ERC vs. LRC

Processor 1
- Acq(L)
- W(a)1
- Rel(L)

Processor 2
- Acq(L)
- R(a)1
- R(b)2
- a = 1

Processor 3
- Acq(L)
- W(b)2
- Rel(L)
- b = 2

How do we know these?
Vector Timestamp & Interval

Processor 1
Acq(L) W(a)1 Rel(L)
(1,0,0) (2,0,0)
{(1,0,0),(2,0,0)}

Processor 2
Acq(L) R(a)1 R(b)2
(2,1,2)

Processor 3
Acq(L) W(b)2 Rel(L)
(2,0,1) (2,0,2)
Write notice

Processor 1

Acq(L) W(a)1 Rel(L)
(1,0,0) (2,0,0)

When Acquiring Lock

Processor 2

Processor 3

Acq(L)
(0,0,0)
Write notice

Processor 1
Acq(L) W(a)1 Rel(L)
(1,0,0) (2,0,0)

Processor 2

Processor 3
Acq(L)
(0,0,0)

“I’m (0,0,0)”
Write notice

Processor 1

- Acq(L)
- W(a)1
- Rel(L)

(1,0,0) (2,0,0)

(2,0,0), (0,0,0)...

Stamps for processor 1 is different

Processor 2

Processor 3

- Acq(L)

(0,0,0)
Write notice

Processor 1

Acq(L)  W(a)1 Rel(L)
(1,0,0)  (2,0,0)

"I modified a at {(1,0,0), (2,0,0)}"

Processor 2

Processor 3

Acq(L)
(0,0,0)
Write notice

Processor 1

Acq(L)  W(a)1 Rel(L)

(1,0,0)  (2,0,0)

"Processor 1 modified a at \{(1,0,0),(2,0,0)\}"

Processor 2

Acq(L) W(b)2 Rel(L)

(2,0,1)  (2,0,2)

Processor 3

Acq(L)  R(a)1 R(b)2
Write notice

Processor 1

\[ \text{Acq(L)} \quad \text{W(a)}1 \text{ Rel(L)} \]

(1,0,0) \quad (2,0,0)

Processor 2

\[ \text{Acq(L)} \quad \text{W(b)}2 \text{ Rel(L)} \]

Processor 3

"Processor 1 modified a at \{(1,0,0),(2,0,0)\}""I’m (0,0,0)"

(2,0,1) \quad (2,0,2)
Write notice

Processor 1

Acq(L) W(a)1 Rel(L)

(1,0,0) (2,0,0)

"Processor 1 modified a at \{(1,0,0),(2,0,0)\}"

Processor 2

Acq(L)

Processor 3

Acq(L) W(b)2 Rel(L)

(2,0,1) (2,0,2)

(2,0,2), (0,0,0)...
Stamps for processor 1, 3 is different
Write notice

Processor 1

Acq(L) W(a)1 Rel(L)

(1,0,0) (2,0,0)

"Processor 1 modified a at \{(1,0,0), (2,0,0)\}"

Processor 2

Acq(L)

"Processor 1 modified a at \{(1,0,0), (2,0,0)\}"

"I modified b at \{(2,0,1), (2,0,2)\}"

Processor 3

Acq(L) W(b)2 Rel(L)

(2,0,1) (2,0,2)
Write notice

Processor 1

Acq(L) W(a)1 Rel(L)

(1,0,0) (2,0,0)

“Processor 1 modified a at {(1,0,0),(2,0,0)}”

“Processor 3 modified b at {(2,0,1), (2,0,2)}”

Processor 2

Acq(L)

Processor 3

Acq(L) W(b)2 Rel(L)

(2,0,1) (2,0,2)
Write notice

Processor 1

Acq(L) → W(a) → Rel(L)

(1,0,0) → (2,0,0)

“Processor 1 modified a at {(1,0,0),(2,0,0)}”

Processor 2

Acq(L) → R(a)

Processor 3

Acq(L) → W(b) → Rel(L)

(2,0,1) → (2,0,2)

“Processor 3 modified b at {(2,0,1), (2,0,2)}”
Write notice

Processor 1

Acq(L)  W(a)1  Rel(L)
(1,0,0)   (2,0,0)

“Processor 1 modified a at {(1,0,0), (2,0,0)}”
“Processor 3 modified b at {(2,0,1), (2,0,2)}”

Processor 2

Acq(L)  R(a)

Processor 3

Acq(L)  W(b)2  Rel(L)
(2,0,1)   (2,0,2)

a = 1
Write notice

Processor 1

Acq(L) W(a)1 Rel(L)

(1,0,0) (2,0,0)

“Processor 1 modified a at \{(1,0,0),(2,0,0)\}”

“Processor 3 modified b at \{(2,0,1),(2,0,2)\}”

Processor 2

Acq(L) R(a)1 R(b)

Processor 3

Acq(L) W(b)2 Rel(L)

(2,0,1) (2,0,2)
Write notice

“Processor 1 modified a at \{(1,0,0),(2,0,0)\}”
“Processor 3 modified b at \{(2,0,1), (2,0,2)\}”

b = 2
False sharing problem

• Modification information is handled in a unit of page
• What happens when variable a and b is in the same page?
Review

Processor 1: Acq(L) → W(a) → Rel(L)

Processor 2: Acq(L) → R(a) → R(b)

Processor 3: Acq(L) → W(b) → Rel(L)
Review

Processor 1

Acq(L)  W(a)1 Rel(L)

Processor 2

Acq(L)  R(a)1R(b)2

Processor 3

Acq(L)  W(b)2 Rel(L)

get Page X
Review

Processor 1

Acq(L)  W(a)1  Rel(L)

Processor 2

Acq(L)  R(a)1  R(b)2

Processor 3

Acq(L)  W(b)2  Rel(L)

get Page X

No problem!
False sharing example

Processor 1

Acq(L1) W(a)1 Rel(L1)

Processor 2

Acq(L1) Acq(L2) R(a)1 R(b)

Processor 3

Acq(L2) W(b)2 Rel(L2)
False sharing example

Processor 1
\[\text{Acq}(L1) \ \text{W}(a)1 \ \text{Rel}(L1)\]

Processor 2
\[\text{Acq}(L1) \ \text{Acq}(L2) \ \text{R}(a)1\text{R}(b)\]

Processor 3
\[\text{Acq}(L2) \ \text{W}(b)2 \ \text{Rel}(L2)\]

get Page X

get Page X

???
Multiple Writer Protocol

• Main idea : Twin & Diff
  • Twin : Before writing something, preserve the original page by copying it
  • Diff : When sending modifications, just send the modified region
Multiple Writer Protocol

Processor 1

Acq(L1) W(a)1 Rel(L1)

Processor 2

Acq(L1) Acq(L2) R(a)1 R(b)

Processor 3

Acq(L2) W(b)2 Rel(L2)
Multiple Writer Protocol

Processor 1: Acq(L1) W(a)1 Rel(L1)

Processor 2: Acq(L1) Acq(L2) R(a)1 R(b)

Processor 3: Acq(L2) W(b)2 Rel(L2)
Multiple Writer Protocol

Processor 1
Acq(L1) W(a)1 Rel(L1)

Processor 2
Acq(L1) Acq(L2) R(a)1R(b)

Processor 3
Acq(L2) W(b)2 Rel(L2)
Multiple Writer Protocol

Processor 1

- Acq(L1) W(a)1 Rel(L1)

Processor 2

- Acq(L1) Acq(L2) R(a)1 R(b)  
  Lazy diff creation

Processor 3

- Acq(L2) W(b)2 Rel(L2)
Multiple Writer Protocol

Processor 1

Acq(L1) W(a)1 Rel(L1)

Processor 2

Acq(L2) W(b)2 Rel(L2)

Processor 3

Acq(L1) Acq(L2) R(a)1 R(b)
Multiple Writer Protocol

Processor 1

Acq(L1) W(a)1 Rel(L1)

Processor 2

Acq(L1) Acq(L2) R(a)1 R(b)

Processor 3

Acq(L2) W(b)2 Rel(L2)
Implementation

• Data structure
Implementation

• Lock management
  • Assigned to all processors in a round-robin manner
  • Support barrier synchronization

• Garbage Collection
  • Use user-level garbage collector

• Portable implementation
  • Uses Unix standard libraries
  • RPC, memory management, etc.
Evaluation

• 8 DECstation-5000/240's running Ultrix V4.3
• Fore ATM Interface & Switch
  • 100-Mbps
• Ethernet Interface & Switch
  • 10-Mbps
• 5 applications
  • Water
  • Jacobi
  • TSP
  • Quicksort
  • ILINK
Evaluation

![Graph showing speedup against processors with different benchmarks and execution statistics]

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Jacobi</th>
<th>TSP</th>
<th>Quicksort</th>
<th>ILINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>343 mols</td>
<td>2000x1000 floats</td>
<td>19-city tour</td>
<td>250000 integers</td>
<td>CLP</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>13.0</td>
<td>32.0</td>
<td>43.8</td>
<td>13.1</td>
<td>1113</td>
</tr>
<tr>
<td>Barriers/sec</td>
<td>2.5</td>
<td>6.3</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>Locks/sec</td>
<td>582.1</td>
<td>0</td>
<td>16.1</td>
<td>53.9</td>
<td>0</td>
</tr>
<tr>
<td>Msgs/sec</td>
<td>2238</td>
<td>334</td>
<td>404</td>
<td>703</td>
<td>456</td>
</tr>
<tr>
<td>Kbytes/sec</td>
<td>798</td>
<td>415</td>
<td>121</td>
<td>788</td>
<td>164</td>
</tr>
</tbody>
</table>

**Figure 4** Execution Statistics for an 8-Processor Run on TreadMarks
Evaluation

Figure 12  Diff Creation Rate (diffs/sec)