Memory Coherence in Shared Virtual Memory Systems

Kai Li
Princeton University

Paul Hudak
Yale University

TOCS, 1989

2006-21163 김신규
Introduction

- The shared virtual memory provides a virtual address space that is shared among all processors in a loosely coupled distributed-memory multiprocessor system.

- The shared virtual memory not only “pages” data between physical memories and disks but it also “pages” data between the physical memories of the individual processors.

- Difficulty
  - The memory coherence problem
Contents

- Shared Virtual Memory
- Memory Coherence Problem
- Centralized Manager Algorithm
- Distributed Manager Algorithm
  - Fixed / Dynamic
- Experiments
- Conclusions
Shared Virtual Memory

Fig. 1. Shared virtual memory mapping.
Memory Mapping Managers

- Roles
  1. Implement the mapping between local memories and the shared virtual memory address space.
  2. Keep the address space coherent at all times.

- The memory mapping manager views its local memory as a large cache of the shared virtual memory address space for its associated processor.
  - A memory reference causes a page fault when the page containing the memory location is not in a processor’s current physical memory. When this happens, the memory mapping manager retrieves the page from either disk or memory of another processor.
Memory Coherence Problem

- Coherent
  - The value returned by a read operation is always the same as the value written by the most recent write operation to the same address.

- Two design choices
  - The granularity of the memory units (Page size).
  - The strategy for maintaining coherence.
Granularity

- In a typical loosely coupled multiprocessor, sending large packets of data is not much more expensive than sending small ones.

- The larger the memory unit, the greater the chance for contention.

- Use the page size of conventional virtual memory implementations (1KB).
## Table I. Spectrum of Solutions to the Memory Coherence Problem

<table>
<thead>
<tr>
<th>Page synchronization method</th>
<th>Page ownership strategy</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed</td>
<td>Centralized manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fixed</td>
</tr>
<tr>
<td>Invalidation</td>
<td>Not allowed</td>
<td>Okay</td>
</tr>
<tr>
<td>Write-broadcast</td>
<td>Very expensive</td>
<td>Very expensive</td>
</tr>
</tbody>
</table>
Page Table, Locking, and Invalidation

- **Page Table**
  - *access*: indicates the accessibility to the page
  - *copy set*: contains the processor numbers that have read copies of the page
  - *lock*: synchronizes multiple page faults by different processes on the same processor and synchronizes remote page requests.

- **Invalidate**
  
  ```
  Invalidate ( p, copy_set )
  For i in copy_set DO
    Send an invalidation request to processor i;
  ```
Centralized Manager Algorithm

- The Centralized Manager
  - Resides on a single processor
  - Maintains a table called **Info**
    - One entry for each page

- Each processor
  - Also has a page table called **PTable**.
  - Keeps information about the accessibility of pages on the local processor
Centralized Manager Algorithm

<table>
<thead>
<tr>
<th>Info</th>
<th>PTable</th>
<th>PTable</th>
<th>PTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>access</td>
<td>access</td>
<td>access</td>
</tr>
<tr>
<td>copy set</td>
<td>lock</td>
<td>lock</td>
<td>lock</td>
</tr>
</tbody>
</table>

- **Info**
  - **Owner** - The single processor that owns that page (= The most recent processor to have write access to it)
  - **Copy Set** - All processors that have copies of the page
  - **Lock** - Synchronizes remote page requests

- **PTable**
  - **Access** - Indicates the accessibility to the page
  - **Lock** - Synchronizes multiple page faults on the same processor
Read-Page Fault in P2

**Info**
- **owner**: P3
- **copy set**: P1, P2
- **lock**

**Ptable**
- **access**: R
- **lock**

Request ➔ Confirmation ➔ Ask to send copy to P2 ➔ Send copy
Write-Page Fault in P2

Request

Invalidation

Confirmation

Send copy

P1

Info

<table>
<thead>
<tr>
<th>owner</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy set</td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td></td>
</tr>
</tbody>
</table>

P2

Ptable

<table>
<thead>
<tr>
<th>access</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td></td>
</tr>
</tbody>
</table>

P3

Ptable

<table>
<thead>
<tr>
<th>access</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td></td>
</tr>
</tbody>
</table>

Ask to send page to P2
An Improved Centralized Manager Algorithm

- **Difference:**
  - The synchronization of page ownership has been moved to the individual owners, thus eliminating the confirmation operation to the manager.
  - The locking mechanism on each processor now deals not only with multiple local requests, but also with remote requests.
  - The manager still answers the question of where a page owner is, but it no longer synchronizes requests.
Distributed Manager Algorithm

- In the Centralized Manager Algorithms
  - There is only one manager for the whole shared virtual memory
  - A traffic bottleneck at the manager may occur as \( N \) becomes large and there are many page faults

- Several ways of distributing the managerial task among the individual processors
  - Fixed/Dynamic
In a fixed distributed manager scheme
- A predetermined subset of the pages to manage
- Choosing an appropriate mapping from pages to processors is difficult → So, dynamic!

\[ H(p) = p \mod N, \ p \in I, \ N \text{ is } \# \text{ of processors} \]

\[ H(p) = (p/s) \mod N, \ s \text{ is } \# \text{ of pages per segment} \]
Broadcast Distributed Manager Algorithm

- Each processor manages precisely those pages that it owns.
- Faulting processors send broadcasts into the network to find the true owner of a page.
- Owner table is eliminated completely, and the information of ownership is stored in each processor’s $PTable$. 

```
<table>
<thead>
<tr>
<th>Owner</th>
<th>Ptable access</th>
<th>lock</th>
<th>copy set</th>
<th>owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy set</td>
<td>access</td>
<td>lock</td>
<td>copy set</td>
<td>owner</td>
</tr>
<tr>
<td>lock</td>
<td>access</td>
<td>lock</td>
<td>copy set</td>
<td>owner</td>
</tr>
<tr>
<td>owner</td>
<td>access</td>
<td>lock</td>
<td>copy set</td>
<td>owner</td>
</tr>
</tbody>
</table>
```
Broadcast Distributed Manager Algorithm

- **Read Fault**
  1. The faulting processor P sends a broadcast read request.
  2. The true owner of the page responds by adding P to the page’s copy set field.
  3. Sending a copy of the page P.

- **Write Fault**
  1. Faulting processor sends a broadcast write request.
  2. The true owner of the page gives up ownership and sends back the page and its copy set.
  3. The requesting processor receives the page and the copy set, it invalidates all copies.
Dynamic Distributed Manager Algorithm

- Keeping track of the ownership of all pages in each processor’s local PTable.
  → probOwner

- Owner → probOwner
  - The true owner or the “Probable” owner

![Diagram]

P1 request P2 forward P3 ...... Pn

probable owner probable owner true owner

page fault
Read-Page Fault in P3

- probOwner
- copyset
- W: writable page
- R: readable page

P3 -> P1: forward request to P1
P1 -> P2: send the page and the copyset \{P1\}
P2 -> P1: ask P2 for read access
P1 -> R: forward request to P1

P3 -> P2: ask P3 for read access
P2 -> P3: send the page and the copyset \{P1\}

Diagram shows the flow of requests and responses involving P1, P2, and P3.
Write-Page Fault in P2

P2

P3

P1

probOwner
copyset
W writable page
R readable page

W ask P3 for write access
W send page and copyset {P1}

R read/lock page
R read/lock page
Two Critical Questions

- Whether forwarding requests eventually arrive at the true owner?
  - Theorem 1. A page fault on any processor reaches the true owner of the page using at most N-1 forwarding request messages

- How many messages are required?
  - Theorem 2. The worst-case number of messages for locating owner of a single page K times is O(N+KlogN)
Using Fewer Broadcasts

- An improvement by using fewer broadcasts
  - The possibility of further improving the algorithm by enforcing a broadcast message after every M page faults to a page.
  - After a Broadcast Request or a Broadcast Invalidation, all processors know the true owner of a page.
  - A counter is needed in each entry of the page table and maintained by its owner.
Distribution of Copy Sets

- The location of the copy set is unimportant
  - Copy set data is distributed and stored as a tree
Distribution of Copy Sets

- Improves system performance for the architectures that do not have a broadcast facility in two important ways.
  - “Divide and Conquer” effect.
    - The propagation of invalidation messages is usually faster.
    - If the copy set tree is perfectly balanced, the invalidation process takes time proportional to \( \log m \) for \( m \) read copies.
  - A read fault now only needs to find a single processor that holds a copy of the page.
Experiments

- Four Parallel Programs
  - Parallel Jacobi program for solving three dimensional partial differential equations (PDEs)
  - Parallel Sorting
  - Parallel matrix Multiply
  - Parallel dot-product program
Results

• 3-D PDE

Fig. 5. Speedups of a 3-D PDE where $n = 50^3$.  

Fig. 7. Speedups of a 3-D PDE where $n = 40^3$.  

Results

Fig. 6. Disk paging on one processor and two processors.
Results

- Parallel merge-split sort
  - Ideal: the costs of all memory references are the same

Fig. 8. Speedup of the merge-split sort.
Results

• Dot-product program

Fig. 9. Speedup of the dot-product program.
Results

• Matrix multiplication program

Fig. 10. Speedup of the matrix multiplication program.
Comparison between Algorithms

![Graph showing comparison between algorithms]

Number of requests vs. Number of processors

- read faults
- write faults
- fixed dist.
- impr. cent.
- dyn. dist.

Fig. 11. Forwarding requests.
Conclusion

- Solving memory coherence problem
  - Centralized manager
  - Distributed manager
    - Fixed / Dynamic

- In general, dynamic distributed manager algorithms perform better than other methods

- This paper gives possibility of using a shared virtual memory system to construct a large-scale shared memory multiprocessor system