Memory Coherence in Shared Virtual Memory System

ACM Transactions on Computer Science (TOCS), 1989

KAI LI Princeton University

PAUL HUDAK Yale University

2009-20811 박정호
Contents

• Introduction
• Shared Virtual Memory
  – Memory Mapping manager
• Memory Coherence Problem
• Solution to the memory coherence problem
  – Centralized Managed Algorithm
  – Distributed Managed Algorithm
• Experiments
• Conclusion
Introduction

• Shared virtual memory
  – Provides a virtual address space that is shared among all processors
  – Applications can use it just as they do a traditional virtual memory
  – Not only “pages” data between physical memories and disks
    • Conventional virtual memory system
  – But also “pages” data between the physical memories of the individual processors
    • Data can naturally migrate between processors on demand

• Memory coherence problem for a shared virtual memory
  – Main difficulty in building a shared virtual memory
  – This paper concentrate on the memory coherence problem
Shared Virtual Memory

- A single address space shared by a number of processors
- Address space is partitioned into pages
Shared Virtual Memory (cont’d)

• Memory mapping managers
  – Mapping between local memories and the shared virtual memory
  – Keeping the address space 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
  – Views its local memory as a large cache

• Key goals
  – To allow processes of a program to execute on different processors in parallel
Memory Coherence Problem

• Memory 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 for build a shared virtual memory system
  – Granularity of the memory units (“page sizes”)
    • Sending large packets
      – is not much more expensive than small one
      – is the greater chance for contention than small one
    • Right size is clearly application dependent
      – 1K bytes is suitable with respect to contention and communication overhead
  – Strategy for maintaining coherence
    • Page synchronization and page ownership
Memory Coherence Problem (cont’d)

• Page synchronization
  – Invalidation
    • Invalidates all copies of the page when write page fault occurs
  – Write-broadcast
    • Writes to all copies of the page when write page fault occurs

• Page Ownership
  – Fixed
    • A page is always owned by the same processor
  – Dynamic
    • Centralized manager
    • Distributed manager
      – Fixed
      – Dynamic
## Memory Coherence Problem (cont’d)

<table>
<thead>
<tr>
<th>Page synchronization method</th>
<th>Page ownership strategy</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Centralized manager</td>
</tr>
<tr>
<td>Invalidation</td>
<td>Fixed</td>
<td>Not allowed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write-broadcast</td>
<td>Very expensive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Memory Coherence Problem (cont’d)

- Page Table
  - Access: indicates the accessibility to the page (nil/read/write)
  - Copy set: contains the processor number 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
  - A primitive that invalidate copies of a page

```
invalidate ( p, copy_set )
for i in copy_set DO
  send an invalidation request to processor i;
```
Centralized Manager Algorithms

- The centralized manager
  - Resides on a single processor
  - Maintains a table called “info” per page
    - Owner: owner that page (= the most recent processor to have write access to it)
    - Copy set: lists all processors that have copies of the page
    - Lock: used for synchronizing request to the page

- Each processors
  - Maintains a page table called “PTable” per page
    - Access, Lock
Centralized Manager Algorithms (cont’d)

Algorithm 1: MonitorCentralManager

Read fault handler:

\[
\text{Lock}(\text{PTable}[\ p].\text{lock});
\]

IF I am manager THEN BEGIN

\[
\text{Lock}(\text{Info}[\ p].\text{lock});
\text{Info}[\ p].\text{copyset}
\]
\[
:= \text{Info}[\ p].\text{copyset} \cup \{\text{ManagerNode}\};
\]
\[
\text{receive page p from Info[ p].owner};
\]
\[
\text{Unlock}(\text{Info}[\ p].\text{lock});
\]
END;
ELSE BEGIN

\[
\text{ask manager for read access to p and a copy of p;}
\]
\[
\text{receive p;}
\]
\[
\text{send confirmation to manager;}
\]
END;

\[
\text{PTable}[\ p].\text{access} := \text{read};
\]
\[
\text{Unlock}(\text{PTable}[\ p].\text{lock});
\]

Read server:

\[
\text{Lock}(\text{PTable}[\ p].\text{lock});
\]

IF I am owner THEN BEGIN

\[
\text{PTable}[\ p].\text{access} := \text{read};
\]
\[
\text{send copy of p;}
\]
END;
\[
\text{Unlock}(\text{PTable}[\ p].\text{lock});
\]

IF I am manager THEN BEGIN

\[
\text{Lock}(\text{Info}[\ p].\text{lock});
\text{Info}[\ p].\text{copyset}
\]
\[
:= \text{Info}[\ p].\text{copyset} \cup \{\text{RequestNode}\};
\]
\[
\text{ask Info[ p].owner to send copy of p to RequestNode;}
\]
\[
\text{receive confirmation from RequestNode;}
\]
\[
\text{Unlock}(\text{Info}[\ p].\text{lock});
\]
END;

Write fault handler:

\[
\text{Lock}(\text{PTable}[\ p].\text{lock});
\]

IF I am manager THEN BEGIN

\[
\text{Lock}(\text{Info}[\ p].\text{lock});
\]
\[
\text{Invalidate}(\ p, \text{Info}[\ p].\text{copyset});
\]
\[
\text{Info}[\ p].\text{copyset} := \{};
\]
\[
\text{Unlock}(\text{Info}[\ p].\text{lock});
\]
END;
ELSE BEGIN

\[
\text{ask manager for write access to p;}
\]
\[
\text{receive p;}
\]
\[
\text{send confirmation to manager;}
\]
END;

\[
\text{PTable}[\ p].\text{access} := \text{write};
\]
\[
\text{Unlock}(\text{PTable}[\ p].\text{lock});
\]

Write server:

\[
\text{Lock}(\text{PTable}[\ p].\text{lock});
\]

IF I am owner THEN BEGIN

\[
\text{send copy of p;}
\]
\[
\text{PTable}[\ p].\text{access} := \text{nil};
\]
END;
\[
\text{Unlock}(\text{PTable}[\ p].\text{lock});
\]

IF I am manager THEN BEGIN

\[
\text{Lock}(\text{Info}[\ p].\text{lock});
\]
\[
\text{Invalidate}(\ p, \text{Info}[\ p].\text{copyset});
\]
\[
\text{Info}[\ p].\text{copyset} := \{};
\]
\[
\text{ask Info[ p].owner to send p to RequestNode;}
\]
\[
\text{receive confirmation from RequestNode;}
\]
\[
\text{Unlock}(\text{Info}[\ p].\text{lock});
\]
END;
Centralized Manager Algorithms (cont’d)

• e.g. [Read page fault in P2]

<table>
<thead>
<tr>
<th>Info</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>P3</td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1 }</td>
</tr>
<tr>
<td>lock</td>
<td>0</td>
</tr>
</tbody>
</table>
Centralized Manager Algorithms (cont’d)

- e.g. [Read page fault in P2]

1. Request

P1, Centralized Manager

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

P2

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

P3

<table>
<thead>
<tr>
<th>PTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
</tr>
<tr>
<td>lock</td>
</tr>
</tbody>
</table>
Centralized Manager Algorithms (cont’d)

- e.g. [Read page fault in P2]

1. Request

P1, Centralized Manager

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

<table>
<thead>
<tr>
<th>P2 PTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
</tr>
<tr>
<td>lock</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P3 PTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
</tr>
<tr>
<td>lock</td>
</tr>
</tbody>
</table>
Centralized Manager Algorithms (cont’d)

- e.g. [Read page fault in P2]

1. Request
2. Ask to send copy to P2
Centralized Manager Algorithms (cont’d)

- e.g. [Read page fault in P2]

1. Request
2. Ask to send copy to P2

P2

PTable
access
lock 1

P1, Centralized Manager

Info

| owner | P3 |
| copy set | { P1, P2 } |
| lock | 1 |

P3

PTable
access R
lock 1

Memory Coherence in Shared Virtual Memory System
Centralized Manager Algorithms (cont’d)

• e.g. [Read page fault in P2]

1. Request

2. Ask to send copy to P2

3. Send copy

P1, Centralized Manager

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

P3

| PTable |
| access | R |
| lock | 1 |
Centralized Manager Algorithms (cont’d)

- e.g. [Read page fault in P2]

P1, Centralized Manager

1. Request
2. Ask to send copy to P2
3. Send copy
4. Confirmation

P2

P3

PTable
access R
lock 0

PTable
access R
lock 1

owner P3
copy set { P1, P2 }
lock 1
Centralized Manager Algorithms (cont’d)

- e.g. [Read page fault in P2]

1. Request
2. Ask to send copy to P2
3. Send copy
4. Confirmation

P1, Centralized Manager

P2

P3

PTable
access  R
lock    0

Info
owner  P3
copy set { P1, P2 }
lock    0

Memory Coherence in Shared Virtual Memory System
Centralized Manager Algorithms (cont’d)

- e.g. [Write page fault in P2]

```
P1, Centralized Manager
```

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

```
P2
```

```
PTable
```

```
access  R
lock    0
```

```
P3
```

```
PTable
```

```
access  R
lock    0
```
Centralized Manager Algorithms (cont’d)

• e.g. [Write page fault in P2]

1. Write Request
Centralized Manager Algorithms (cont’d)

- e.g. [Write page fault in P2]

1. Write Request
2. Invalidate of copy set
Centralized Manager Algorithms (cont’d)

- e.g. [Write page fault in P2]

1. Write Request
2. Invalidate of copy set
3. Ask to send page to P2
Centralized Manager Algorithms (cont’d)

- e.g. [Write page fault in P2]

1. Write Request
2. Invalidate of copy set
3. Ask to send page to P2
4. Send page to P2
Centralized Manager Algorithms (cont’d)

- e.g. [Write page fault in P2]
Centralized Manager Algorithms (cont’d)

- An Improved Centralized Manager Algorithm
  - The synchronization of page ownership move to the individual owners
    - Eliminating the confirmation message to the manager
  - The locking mechanism on each processor now deals not only with multiple local requests, but also with remote requests

- But, for large N there still might be a bottleneck at the manager processor because it must respond to every page fault
**Distributed Manager Algorithms**

- **A Fixed Distributed Manager Algorithm**
  - Every processor is given a predetermined subset of the pages to manager
  - e.g.,
    - \( H(p) = p \mod N \), \( N \) is \# of processors
    - \( H(p) = (p/s) \mod N \), \( s \) is \# of pages per segment (e.g., \( s = 3 \))

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

- Proceeds as in the centralized manager algorithm
- The fixed distributed manager algorithm is superior to the centralized manager algorithms when a parallel program exhibits a high rate of page faults
Distributed Manager Algorithms (cont’d)

- A Broadcast Distributed Manager Algorithm
  - Each processor manages precisely those pages that it owns
  - Faulting processor send broadcasts into the network to find the true owner of a page
  - The Owner table is eliminated
  - Information of ownership is stored in each processor’s PTable
Distributed Manager Algorithms (cont’d)

- e.g., [Read page fault in P2]

<table>
<thead>
<tr>
<th>P1</th>
<th>Access</th>
<th>Lock</th>
<th>Copy Set</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td>P3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>P3</th>
<th>PTable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>access</td>
</tr>
<tr>
<td></td>
<td>lock</td>
</tr>
<tr>
<td></td>
<td>copy set</td>
</tr>
<tr>
<td></td>
<td>owner</td>
</tr>
</tbody>
</table>
Distributed Manager Algorithms (cont’d)

- e.g., [Read page fault in P2]

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>access</td>
<td></td>
<td>copy set</td>
<td>owner</td>
</tr>
<tr>
<td>P3</td>
<td>access</td>
<td></td>
<td>copy set</td>
<td>owner</td>
</tr>
</tbody>
</table>

1. Broadcast read request

Memory Coherence in Shared Virtual Memory System
Distributed Manager Algorithms (cont’d)

- e.g., [Read page fault in P2]

1. Broadcast read request
2. Send copy of p

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td>P3</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>access</td>
<td></td>
<td></td>
<td>P3</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td>(P1, P2)</td>
<td></td>
</tr>
</tbody>
</table>
Distributed Manager Algorithms (cont’d)

• A Dynamic Distributed Manager Algorithm
  – The heart of this algorithm is keeping track of the ownership of all pages in each processors’ local Ptable
    • There is no fixed owner or manager
    • Owner field -> probOwner field
    • The information that it contains is just a hint
    • When a processor has a page fault, it sends request to the processor indicated by the probOwner field
      – If that processor is the true owner, it proceeds as in the centralized manager algorithm
      – Else, forwards the request to the processor indicated by its probOwner field
Distributed Manager Algorithms (cont’d)

• e.g., [Read page fault in P2]

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PTable</td>
<td>access</td>
<td>lock</td>
<td>probOwner</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td>P3</td>
</tr>
<tr>
<td>access</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td>P1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1 }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>probOwner</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PTable</td>
<td>access</td>
<td>lock</td>
<td>probOwner</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td>P1</td>
</tr>
<tr>
<td>access</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td>P1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1 }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>probOwner</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P3</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PTable</td>
<td>access</td>
<td>lock</td>
<td>probOwner</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td>P3</td>
</tr>
<tr>
<td>access</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1 }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>probOwner</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distributed Manager Algorithms (cont’d)

- e.g., [Read page fault in P2]

1. read request

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>probOwner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ P1 }</td>
<td>P3</td>
</tr>
</tbody>
</table>

P1

P2

P3
Distributed Manager Algorithms (cont’d)

- e.g., [Read page fault in P2]

1. read request
2. forward request
Distributed Manager Algorithms (cont’d)

- e.g., [Read page fault in P2]

1. read request
2. forward request
3. Send copy of p and copy set
Distributed Manager Algorithms (cont’d)

- e.g. [Write page fault in P1]

```
<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1 }</td>
<td></td>
</tr>
<tr>
<td>probOwner</td>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1, P3 }</td>
<td></td>
</tr>
<tr>
<td>probOwner</td>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy set</td>
<td>{ P1, P3 }</td>
<td></td>
</tr>
<tr>
<td>probOwner</td>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>
```
Distributed Manager Algorithms (cont’d)

- e.g. [Write page fault in P1]

1. write request
Distributed Manager Algorithms (cont’d)

- e.g. [Write page fault in P1]

1. write request

2. Send p and copy set

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>probOwner</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R</td>
<td></td>
<td>{ P1 }</td>
<td>P2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>probOwner</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td>{ P1, P3 }</td>
<td>P2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>probOwner</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>R</td>
<td></td>
<td>{ P1, P3 }</td>
<td>P2</td>
</tr>
</tbody>
</table>
Distributed Manager Algorithms (cont’d)

- e.g. [Write page fault in P1]

1. Write request
2. Send p and copy set
3. Invalidate in copy set

<table>
<thead>
<tr>
<th>PTable</th>
<th>access</th>
<th>lock</th>
<th>copy set</th>
<th>probOwner</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>access</td>
<td></td>
<td>copy set</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>access</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTable</td>
<td>access</td>
<td>lock</td>
<td>copy set</td>
<td>probOwner</td>
</tr>
<tr>
<td></td>
<td>access</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTable</td>
<td>access</td>
<td>lock</td>
<td>copy set</td>
<td>probOwner</td>
</tr>
<tr>
<td></td>
<td>access</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distributed Manager Algorithms (cont’d)

- e.g. [Write page fault in P1]

1. Write request
2. Send p and copy set
3. Invalidate in copy set
Distributed Manager Algorithms (cont’d)

• Distribution of Copy Set
  – Copy set data is distributed and stored as a tree

  – Improves system performance
  • “Divide and conquer” effect
    – Balanced tree -> invalidation process takes log m for m read copies
  • A read fault only needs to find a single processor that holds a copy of the page (not necessarily the owner)
Experiments

• Experimental Setup
  – Implement a prototype shared virtual memory system called IVY (Integrated shared Virtual memory at Yale)
    • Implemented on top of a modified Aegis operating system of Apollo DOMAIN computer system
    • IVY can be used to run parallel programs on any number of processors on an Apollo ring network

  – Parallel Programs
    • Parallel Jacobi program
      – Solving three dimensional PDEs (partial differential equations)
    • Parallel sorting
    • Parallel matrix multiplication
    • Parallel dot-product
Experiments (cont’d)

- Speed up of 3D-PDE, where \( n = 50^3 \)

- Super-linear speed up
  - Data structure for the problem is greater than the size of physical memory on a single processor
Experiments (cont’d)

- Speed up of 3-D PDE, where $n=40^3$

![Graph showing speedup vs. number of processors for an ideal solution and an experimental result.](image)

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

Memory Coherence in Shared Virtual Memory System
Experiments (cont’d)

- Speedup of the merge-split sort, where 200K elements
  - Ideal solution
    - Even with no communication costs, merge-split sort does not yield linear speed up
    - The costs of all memory references are the same

![Graph showing speedup over number of processors with Ideal Solution and Experimental Result]

Fig. 8. Speedup of the merge-split sort.
Experiments (cont’d)

- Speed up of parallel dot-product where each vector has 128K elements

- Shared virtual memory system is not likely to provide speed up
  - Ratio of the communication cost to the computation cost is large
Experiments (cont’d)

- Speed up of matrix multiplication, where 128 x 128 square matrix

Fig. 10. Speedup of the matrix multiplication program.
Experiments (cont’d)

• Compare to memory coherence algorithms
  – All algorithms have similar number of page faults
  – Number or requests of fixed distributed manager algorithm is similar to improved centralized manager algorithm

• Both algorithm need a request to locate the owner of the page for almost every page fault that occurs on a non-manager processor
Conclusion

- This paper studied two general classes of algorithms for solving the memory coherence problem
  - Centralized manager
    - Straightforward and easy to implement
    - Traffic bottleneck at the central manager
  - Distributed Manager
    - Fixed dist. Manager
      - Alleviates the bottleneck
      - On average, still need to spend about two message to locate an owner
    - Dynamic dist. Manager
      - The most desirable overall features
- This paper gives possibility of using a shared virtual memory system to construct a large-scale shared memory multiprocessor system
Q & A

Thank you!