Distributed Shared Memory Integration

Mordechai Geva and Yair Wiseman
Bar Ilan University
Israel
DSM Portability

- The DSM performance issues have been preferred over portability issues.
- The APIs of DSMs fit only specific systems.
  - Migration might require a significant revision.
- Programs for SMPs should be rewritten if a scale up for a cluster is needed.
- Programmers need to acquire new programming skills for each DSM system.
What do We Aim for?

- A DSM API that will be identical to the standard UNIX API.
- A DSM that is not bundled within other IPC elements.
  - As an expansion, we require that all IPC elements will not be bundled within other IPC elements.
- The very same program can be compiled on a PC, an SMP or a cluster with even not a single change.
Approaches of DSM systems

- Virtual DSM
- Object DSM
- Thread Migration
- Compiler supported DSM
Virtual DSM

- Each page on each node can be *available* or *unavailable*.
- A process may handle only *available* pages.
- If a process encounters an *unavailable* page, the page will be fetched to the node and its status will become *available*.
- A page can be *available* only at one node.
Object DSM

- The page resides only at one node.
- Each process that has to access the page will connect the page’s host.
- Some systems allows the page having a replicas on one or more machine.
  - In such cases a writing should be done in all the replicas.
Thread Migrations

- Move the processes or the threads towards the pages, instead of moving the pages towards the processes.
- A process that tries to access a page, will be moved to the page’s host.
Complier Supported DSM

- The program hints the compiler regarding the shared memory access manner.
- For example, hinting that a page is write-shared, will allow multiple threads to change its content without disallowing other threads from doing so.
- This approach is very efficient, but the API becomes very cumbersome.
Portable DSM

- **Portability** - Built as a user level lib.
- **Usability** - Having the same API as in standard UNIX systems.
- **Disjointing** - Creating a portable, stand-alone, multiple-processor IPCs, i.e. not a kernel implementation.
The Implementation

- The implementation was done in the most common approach - Virtual DSM; however it obviously can be implemented by any other approach.

- The well-known “Segmentation Fault” signal was used to check whether the page is available on current host.
Daemons

- **dsmserver** - Running only on the server processor and responsible for:
  - Allocations of page numbers to processes.
  - Saving pages after process termination.

- **mcdaemon** - Running on each processor and responsible for:
  - Transferring a locally *available* pages upon requests.
Handling Segmentation Faults

- When a process accesses an *unavailable* page, UNIX will send the process a “segmentation fault” signal.
- A dedicated signal handler in the library will catch the signal.
- The signal handler checks whether it is a real “segmentation fault” or an *unavailable* page.
- If it is an *unavailable* page, a multicast is sent requesting the page.
- The mcdaemon in the processor that has the page, will send the page and will mark it as an *unavailable* page.
Semaphores

- A supplementary IPC element for the DSM is the Remote Semaphores.
- Similarly to DSM, the Remote Semaphores have two daemons:
  - **Server Daemon** on the server processor that handles the id numbers and the location of the semaphores.
  - **Client Daemon** on each client processor that handles the real semaphores.
Benchmark - Dense Matrix Multiplication

<table>
<thead>
<tr>
<th>Matrix sizes</th>
<th>Single computer</th>
<th>Two computers</th>
<th>Four computers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1</td>
<td>0.517</td>
<td>0.258</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
<td>0.521</td>
<td>0.271</td>
</tr>
<tr>
<td>3072</td>
<td>1</td>
<td>0.527</td>
<td>0.283</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
<td>0.542</td>
<td>0.295</td>
</tr>
</tbody>
</table>

Runtime reduction graph:
- Single computer
- Two computers
- Four computers
Oevrhead

Overhead percentage

Matrix size

Overhead

2p overhead

4p overhead

0.0166667 0.0213319 0.0266627 0.042375

0.0083333 0.0205515 0.0331037 0.0453565

1024 X 2048 X 3072 X 4096 X
Conclusions

- Nowadays DSMs are very sophisticated and complicated; thus can help solve many advanced problems.
- However, the cumbersome API deters programmers from using advanced DSM packages.
- We believe this common UNIX API for DSM can bring the integration of advanced DSM packages into play.