Overcoming the Problems Associated with the Existence of Too Many DSM APIs

Martin Schulz
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Technische Universität München

DSM 2002, Berlin
May 22nd, 2002
Many APIs!

- Large variety of shared memory APIs
  - Includes DSM APIs from various projects

- Same base functionality
  - Global memory allocation
  - Implicit communication via common address space
  - Synchronization to safely share data

- Slight difference in syntax and semantics
  - Task model
  - Issue of memory allocation requests
  - Coherency model
Too Many APIs ?!

- Porting between DSM system difficult
  - Each DSM system has its own application base
  - No automatic reuse of porting knowledge

- First approach: Unified model
  - Propose and lobby for a new API for everybody
  - Difficult/Impossible to get accepted

- Second approach: Hide differences
  - Flexible framework supporting „all“ APIs
  - Retargetable system with multiple faces
  - Minimized porting efforts (port once)
Outline

- The HAMSTER framework
  - Overview
  - Service Architecture
  - Management Modules

- Implementing DSM on top of HAMSTER
  - TreadMarks, HLRC, and JiaJia
  - Implementation
  - Complexity Evaluation

- Status, Outlook, Conclusions
The HAMSTER Approach

- **HAMSTER**: Hybrid-dsm based Adaptive and Modular Shared memory architecture

- Two components
  - Efficient Hybrid-DSM system
  - Shared Memory Service Modules

- Goals
  - Low complex construction of models
  - Enable the use of several APIs
  - „Porting the model instead of the application“
  - Also includes support for several DSM APIs
HAMSTER Overview

Shared Memory application

Shared Memory programming model

DSM system (with SAN support), which serves as efficient base and applications

Base modules for shared memory support

Recipes for shared memory

SCI-VM: Hybrid DSM for SCI clusters

VI-like comm. access through HW-DSM

SAN with HW-DSM

Standalone OS

Linux & WinNT

Cluster built of commodity PC hardware

HAMSTER Overview

Shared Memory application

Shared Memory programming model

DSM system (with SAN support), which serves as efficient base and applications

Base modules for shared memory support

Recipes for shared memory

SCI-VM: Hybrid DSM for SCI clusters

VI-like comm. access through HW-DSM

SAN with HW-DSM

Standalone OS

Linux & WinNT

Cluster built of commodity PC hardware

HAMSTER Overview

Shared Memory application

Shared Memory programming model

DSM system (with SAN support), which serves as efficient base and applications

Base modules for shared memory support

Recipes for shared memory

SCI-VM: Hybrid DSM for SCI clusters

VI-like comm. access through HW-DSM

SAN with HW-DSM

Standalone OS

Linux & WinNT

Cluster built of commodity PC hardware

HAMSTER Overview

Shared Memory application

Shared Memory programming model

DSM system (with SAN support), which serves as efficient base and applications

Base modules for shared memory support

Recipes for shared memory

SCI-VM: Hybrid DSM for SCI clusters

VI-like comm. access through HW-DSM

SAN with HW-DSM

Standalone OS

Linux & WinNT

Cluster built of commodity PC hardware

HAMSTER Overview

Shared Memory application

Shared Memory programming model

DSM system (with SAN support), which serves as efficient base and applications

Base modules for shared memory support

Recipes for shared memory

SCI-VM: Hybrid DSM for SCI clusters

VI-like comm. access through HW-DSM

SAN with HW-DSM

Standalone OS

Linux & WinNT

Cluster built of commodity PC hardware
Service Architecture

- Large range of services
  - Represent common features of shmem models
  - Flexible, large range of parameters

- Encapsulated into orthogonal modules
  - Split into service groups
  - Safe use without side effects

- Implemented on top of single core DSM
  - Responsible for global memory abstraction
  - Hide system specifics
Management Modules

- **Clus.Ctrl.**
  - Task Mgmt.
- **Sync. Mgmt.**
- **Cons. Mgmt.**
- **Mem. Mgmt.**

**SCI-VM: Hybrid DSM for SCI clusters**

**Standalone OS Linux & WinNT**

**NIC driver**

**Cluster built of commodity PC hardware**

**SAN with HW-DSM**

**VI-like comm. access through HW-DSM**

- SAN with HW-DSM
Shared Memory Services

- Memory Management
  - Global memory allocation
  - Specification of data distributions

- Consistency Management
  - Control memory consistency
  - Mechanisms to flush & invalidate state

- Synchronization Management
  - Standard mechanisms – Locks & Barriers
  - Support for further specific constructs
  - Lean and low-level implementations
Task & Cluster Control

- Task management
  - Registration and Deregistration of local threads
  - Activity counter

- Cluster control
  - Team coordination across nodes
  - Simple RPC like messaging support
  - Clean termination

- Result: HAMSTER interface
  - Collection of all services
  - Some general services added / E.g. timing
Implementing Consistency Models

- **Memory Consistency Models**
  - Distinguishing criterion for shared memory APIs
  - DSM systems mostly based on relaxed models

- **Can not be hard-wired in HAMSTER**
  - Need to support as many APIs as possible
  - Task for the programming model layer

- **Separation of Consistency and Synchronization**
  - Separate modules
  - Combination in the programming model layer
  - Allows existing and new, specialized CM
Adding programming models

- Shared Memory application
  - Shared Memory programming model
    - Clus.Ctrl.
    - Task Mgmt.
    - Sync. Mgmt.
    - Cons. Mgmt.
    - Mem. Mgmt.
    - SCI-VM: Hybrid DSM for SCI clusters
  - Standalone OS
    - Linux & WinNT
    - NIC driver
  - Cluster built of commodity PC hardware
  - SAN with HW-DSM
  - VI-like comm. access through HW-DSM
Implementing programming models

- Issues in building programming models
  - Memory consistency model
  - Task structure and initialization
  - Type of allocation policy

- Low complexity
  - Specialized HAMSTER services
  - Only few routines to be implemented from scratch

- Base model: SPMD
  - Native model expected from the DSM
  - Currently provided by SCI-VM
Sample DSM APIs

- Study of three well-known APIs
  - TreadMarks™ (commercial, Rice & KAI/Intel)
  - HLRC (free, Rutgers)
  - JiaJia (free, Chinese Academy of Science)

- Issues
  - API characteristics in terms of allocations
  - Consistency model
  - Implementation aspects
  - Implementation complexity

- Common: static task model
Example 1: TreadMark™ API

- **Characteristics**
  - Local memory allocation (by one thread)
  - Pre-allocated synchronization operations
  - Release Consistency Model (RC)

- **Implementation**
  - Memory allocation and synchronization directly mapped to HAMSTER services
  - Special distribution routine required
    - Implemented via internal shared memory
    - Allocated during programming model setup
  - Automatic initialization for preallocation
Implementing RC

- RC combines consistency with lock operations
  - Guarantee up-to-date memory within critical regions
  - Propagate changes after unlock operations

- Lock/Acquire
  - Perform lock operation (SyncMod)
  - Perform invalidation (ConsMod)

- Unlock/Release
  - Perform flush (ConsMod)
  - Perform unlock (SyncMod)

- Result is (L)RC compliant
Example 2: HLRC API

- **Characteristics**
  - Global memory allocation (SPMD style)
  - Global synchronization allocation
  - Release consistency model (RC)

- **Implementation**
  - Global allocation procedure
    - Designated thread/note performs allocation
    - Internal barrier to ensure completion
    - Distribution of result through internal shared memory
  - Used for both synchronization and memory
  - Consistency model as before
Example 3: JiaJia API

- **Characteristics**
  - Global memory allocation (SPMD style)
    - Includes locality specifications
  - Pre-allocated synchronization operations
  - Scope consistency model (ScC)

- **Implementation**
  - Allocation for memory as on HLRC
  - Synchronization preallocation at startup
  - Locality specifications
    - HAMSTER parameters during memory allocation
Adding Scopes

- Straightforward extension of RC
  - Restrict Acquires and Releases to same scope
  - Often one scope for each lock
  - RC is special case with a single, global scope

- Consistency modul supports scopes
  - Flushes always executed On Release
  - Invalidations only when necessary
    - Invalidations are global
    - Only necessary if no invalidation happened after last Release on same scope
  - HAMSTER allows individual scope allocation
## Implementation Complexity

<table>
<thead>
<tr>
<th>Programming Model</th>
<th># Lines</th>
<th># API calls</th>
<th>Lines/call</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>SPMD</em></td>
<td>502</td>
<td>23</td>
<td>21.8</td>
<td><em>NT &amp; Lin.</em></td>
</tr>
<tr>
<td>TreadMarks™ API</td>
<td>326</td>
<td>13</td>
<td>25.1</td>
<td><em>NT &amp; Lin.</em></td>
</tr>
<tr>
<td>HLRC API*</td>
<td>137</td>
<td>25</td>
<td>5.5</td>
<td><em>NT &amp; Lin.</em></td>
</tr>
<tr>
<td>JiaJia API (subset)*</td>
<td>125</td>
<td>21</td>
<td>5.6</td>
<td><em>NT &amp; Lin.</em></td>
</tr>
</tbody>
</table>

*Based on subset of SPMD

- Low complexity for programming model layer
  - Mostly specialization of HAMSTER services
  - Typical implementation time: ½ day
  - Free add-on: portability between platforms
A Remark on Performance

- HAMSTER relies on a given DSM core
  - All APIs use the same memory abstraction
  - Performance depends on this core
  - Management Layer thin and mainly for setup
  - *Performance Portability*

- Critical point: consistency model
  - In SW-DSM often heavily tuned and integrated
  - Not required for Hybrid-DSM
  - Can be emulated with consistency hooks within the synchronization routines
Status and Outlook

- HAMSTER v0.99 available
  - Based on SCI-VM DSM system
  - Most current version on Linux
    - Earlier versions also on Windows
  - Other programming models include
    - Distributed POSIX and Win32 threads
    - Cray shmem API for one-sided communication

- Next steps
  - Support several different DSM cores
  - Evaluate further programming models
    - E.g. OpenMP
Conclusions

- DSM research mostly focuses on performance
  - Work on protocol details
  - Each system comes with own API
  - Missing portability leads to low acceptance

- HAMSTER aims at overcoming this problem
  - Flexible services on top of common core
  - Enable large range of programming models
  - Easily retargetable system

- Concept shown based on several DSM APIs
  - Including several consistency models
For the curious...

- More on the SMiLE software infrastructure
  Talk at CC-Grid 2002
  Session on Friday 24th, 15:30-17:30

- Web pages:
  http://smile.in.tum.de/
  http://hamster.in.tum.de/

- Email contact:
  schulzm@in.tum.de