SCI Programming

*With a focus on Shared Memory*

Martin Schulz

Lehrstuhl für Rechnertechnik und Rechnerorganisation
Technische Universität München

Trinity College Dublin, Ireland
February 21st/22nd 2002
Overview

- How to make use of SCI
  - User-level communication
  - The SMiLE project at LRR-TUM
- Traditional programming of SCI
  - Raw Shared Memory using the SISCI API
  - Message Passing using SISCI
- HAMSTER: Shared memory for clusters
  - A virtual memory abstraction for NCC-NUMA
  - The role of the programming model
  - Examples of Applications
- Outlook on Lab Session
PC Clustering with SCI

- Very powerful hardware
  - Excellent single node performance
  - Network with low-latency and high bandwidth

- Main advantage of SCI
  - Enables remote memory accesses / HW-DSM
  - Results in NUMA architecture

- Can be used for HPC
  - Question: How to exploit hardware for applications?
  - Important concept: User-level communication
User level communication

- Only setup via kernel
  - Communication direct

- Advantages
  - No OS Overhead
  - No Protocols
  - Direct HW Utilization

- Typical performance
  - Latency < 10 µs
  - Bandwidth > 80 MB/s
    (for 32 bit, 33 MHz PCI)

---

Martin Schulz, Lecture on SCI Shared Memory Programming, Trinity College Dublin, February 21st/22nd 2002
Communication with SCI

- **Export of physical memory**
  - SCI physical address space

- **Mapping into virtual memory**
  - SCI to PCI address space
  - PCI to virtual address space

- **User-level communication**
  - Read/write to mapped segment

- **High performance**
  - No protocol overhead
  - No OS influence
  - Latency: < 2 µs
  - Bandwidth: 89 MB/s (32bit)
SCI Remote Memory Mappings

Node A

Virtual address space on A

PCI addr. on A

Physical mem. on A

CPU MMU

SCI Bridge ATTs

SCI physical address space

Node B

Virtual address space on B

PCI addr. on B

Physical mem. on B

CPU MMU
Questions

- SCI offers flexible NUMA architecture
  - Low-overhead / user-level communication

- Question 1: How to build SCI clusters?
  - Hardware Issues
  - Cluster Configuration and Management

- Question 2: How to exploit SCI?
  - Programming models

- Question 3: How to support the user?
  - Tools & Programming Environments
The SMiLE project

- **Shared Memory in a LAN-like Environment**
  - SCI as the enabling technology
  - Exploiting SCI’s capabilities and opportunities

- **Hardware developments**
  - SMiLE PCI-SCI adapter
  - Hardware monitor to trace remote accesses
  - Based on FPGAs to ensure extensibility

- **Software developments**
  - Comprehensive software infrastructure
  - Support for message passing & shared memory
  - Tool support for programmers
## SMiLE Software Layers

<table>
<thead>
<tr>
<th>Target applications / Test suites</th>
<th>High-level SMiLE</th>
<th>Low-level SMiLE</th>
<th>User/Kernel boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT prot. stack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SISAL on MuSE</td>
<td>SISCI API / Library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSIX threads API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TreadMarks compatible API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM 2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS-lib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCI-Messaging</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCI-VM lib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDIS driver</td>
<td>SCI Drivers &amp; SISCI API</td>
<td>SCI-VM</td>
<td></td>
</tr>
<tr>
<td>SCI-Hardware: SMiLE &amp; Dolphin adapter, HW-Monitor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The SISCI API

- Standard low-level API for SCI-based systems
  - Developed within the SISCI Project
    - Standard Software Infrastructure for SCI
  - Implemented by Dolphin ICS (for many platforms)

- Goal: Provide the full SCI functionality
  - Raw access to SCI’s HW-DSM
  - Secure abstraction of user-level communication
  - Based on the IRM (Interconnect Resource Mng.)
    - Global resource management
    - Fault tolerance & configuration mechanisms
  - Comprehensive user-level API
Use of SISCI API

- Provides raw shared memory functionalities
  - Based on individual shared memory segments
  - No task and/or connection management
  - High complexity due to low-level character

- Application area 1: Middleware
  - E.g. base for efficient message passing layers
  - Hide the low-level character of SISCI

- Application area 2: Special purpose applications
  - Directly use the raw performance of SCI
SISCI availability

- SISCI API directly available from Dolphin
  - [http://www.dolphinics.no/](http://www.dolphinics.no/)
  - Multiplatform support
    - Windows NT 4.0 / 2000
    - Linux (currently for 2.2.x and 2.4.x kernels)
    - Solaris 2.5.1/7/8 (Sparc) & 2.6/7 (X86)
    - Lynx OS 3.01 (x86 & PPC), VxWorks
    - True64 (first version available)
  - Open Source

- Documentation also On-line
  - Complete user-library specification
  - Sample codes
Basic SISCI functionality

- Main functionality:
  - Basic shared segment management
    - Creation of segments
    - Sharing/Identification of segments
    - Mapping of segments

- Support for Memory transfer
  - Sequence control for data integrity
  - Optimized SCI memcpy version

- Direct abstraction of the SCI principle
Advanced SISCI functionality

- Other functionality
  - DMA transfers
  - SCI-based interrupts
  - Store-barriers

- Implicit resource management

- Additional library in SISCI kit
  - sisci_demolib
  - Itself based on SISCI
  - Easier Node/Adapter identification
Allocating and Mapping of Segments

SCICreateSegment
SCIPrepareSegment
SCIMapLocalSegment
SCISetSegmentAvailable

SCICreateSegment
SCIPrepareSegment
SCIMapRemoteSegment

SCIUnMapSegment
SCIDisconnectSegment

Data Transfer

Node A   Node B

SISCI Deficits

- SISCI based on individual segments
  - Individual segments with own address space
  - Placed on a single node
  - No global virtual memory

- No task/thread control, minimal synchronization
  - No full programming environment
  - Has to rebuild over and over again

- No cluster global abstraction
  - Configuration and ID management missing
Message Passing on top of SISCI

- SISAL on MuSE
- SISCO MPI
- SISCO PVM

- AM 2.0
- SS-lib
- CML
- SCI-Messaging

- SCI Drivers & SISCO API

- NDIS driver
- SCI-Hardware: SMiLE & Dolphin adapter, HW-Monitor

- POSIX Riches API

- HAMSTER modules

- User/Kernel boundary

- Target applications / Test suites

- Other prog. models (e.g. visual programs)

- SMiLE Message Passing

- High-level MiLE

- Low-level SMiLE
Messing Principle

- Same principles for all message engines
  - Use of pairwise message buffers
  - Communication based on remote writes

- SMiLE low-level messaging layers
  - Active Messages, User level sockets, CML
  - 80-90% of raw bandwidth, latency < 10\mu s
  - Base for higher-level messaging
Example: Sending from A to B

Node A / Sender
- Send Ringbuffer (mapped)
- End Pointer mapped
- Start Pointer copy
- mapped End Pointer

Node B / Receiver
- Receive Ringbuffer
- Sync
- End pointer
- Start Pointer copy
- mapped Start Pointer

Mapping via SCI
Shared Memory on Clusters

- Problem: memory physically distributed
  - Needed: global virtual memory abstraction
  - Mostly: SW-DSM systems (e.g. Ivy, TreadMarks,...)
    - Mostly page based, using virtual memory traps

- Main deficits
  - Performance problems & complex update protocols
  - Proprietary APIs restrict code portability

- Using SCI to create a more efficient DSM
  - Raw SCI / SISCI is not enough
  - Need for additional software components
HAMSTER

Hybrid DSM based
Adaptive and
Modular
Shared Memory
architectuRe

POSIX threads API
TreadMarks compatible API
HAMSTER modules
SCI-VM

Applications / Test suites

High-level SMiLE
Low-level SMiLE
User/Kernel boundary

SCI-Hardware: SMiLE & Dolphin adapter, HW-Monitor
SCI-VM

SCI Drivers & SISCI API

Target applications / Test suites

SISAL on MuSE
SCI-Messaging

Progr. | SCI-Messaging
NDIS driver | SCI Drivers & SISCI API
Key Ideas of HAMSTER

- Using NUMA properties for DSM
  - But: NUMA only based on physical addresses
  - Need to close the gap to virtual addresses

- Hybrid DSM system
  - Setup & management in software
  - Communication in hardware
  - No protocol overhead
  - Optimal system utilization

- Large variety of shared memory models
  - HAMSTER as common base
HAMSTER overview

Shared Memory application

Shared Memory programming model


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
A global virtual memory abstraction

- SAN with HW-DSM
- Cluster built of commodity PC hardware
- NIC driver
- Standalone OS
  - Linux & WinNT
- SCI-VM: Hybrid DSM for SCI clusters
Global Virtual Memory with SCI

- **SCI Virtual Memory**
  - Flexible, general global memory abstraction
  - One global virtual address space across nodes
  - Total transparency for the user
  - Direct utilization of SCI HW-DSM

- **Hybrid-DSM based system**
  - HW-DSM for implicit communication
  - SW-DSM component for system management

- **Prerequisite for true shared memory models**
  - Bridge the gap between OS instances
SCI-VM design

- Global process abstraction
  - Team on each node host threads
  - Each team needs identical virtual memory view

- Transparent access to complete virtual memory
  - Map local pages conventionally via MMU
  - Map remote pages with the help of SCI
  - Software coordinates these mappings

- Extending the OS’s VM management
  - Fine grain page mappings
  - Access to internal callbacks
SCI-VM mapping scheme
Problem: Caches

- System contains distributed caches
  - Caches can become inconsistent
  - Stale data in cache

- Need to synchronize between caches
  - Hardware solutions: CC-NUMA
  - (Distributed) coherency controller
  - Adds hardware complexity to the system

- SCI clusters implement non-coherent NUMA
  - PCI busses unable to snoop memory traffic
  - Potentially even the more scalable solution
Caches in NUMA Architectures

Global physical address space = HW-DSM

Martin Schulz, Lecture on SCI Shared Memory Programming, Trinity College Dublin, February 21st/22nd 2002
Relaxing the Consistency

- Dealing with NCC-NUMA
  - Novel hybrid cache management
  - Caching in hardware & Control in software

- Enable caching and exploit additional buffers
  - Control with explicit flushes / invalidations
  - Coordination with application requirements

- Relaxed Consistency Models
  - Combine with synchronization constructs
  - Allows formalized consistency constraints
  - Currently implemented: (L)RC & Scope
Release Consistency (RC)

- Classify all accesses

- RC rules
  - Before regular access
    - Complete all Acquire
  - Before Release
    - All reg. accesses completed
    - Sync. accesses strictly ordered

- Explicit signalling
  - Data ready, useful for others ⇒ Release
  - Update required ⇒ Acquire
Programmability

- Direct impact on applications
  - Explicit consistency management

- Combine with constructs for synchronization
  - Acquire & Lock
  - Release & Unlock
  - Acq.+Rel. & Barrier

- Natural extension of lock semantics

- Minimal impact
  - Most „well behaved“ shared memory codes work
Adding 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

Standalone OS Linux & WinNT

SCSI-VM: Hybrid DSM for SCI clusters

NIC driver

Cluster built of commodity PC hardware

VI-like comm. access through HW-DSM

Clus.Ctrl.

Task Mgmt.

Sync. Mgmt.

Cons. Mgmt.

Mem. Mgmt.

SCI-VM: Hybrid DSM for SCI clusters

NIC driver

Cluster built of commodity PC hardware

SAN with HW-DSM

VI-like comm. access through 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

- Current deficits
  - No I/O transparency
Adding programming models

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

- Basis: HAMSTER interface
  - Collection of services exported by modules

- Issues in building programming models
  - Memory consistency model
  - Task structure
  - Initialization

- Low complexity
  - Specialized HAMSTER services
  - Only few routines to be implemented from scratch
# Existing programming models

<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>SPMD</td>
<td>502</td>
<td>23</td>
<td>21.8</td>
<td>NT &amp; Lin.</td>
</tr>
<tr>
<td>SMP/SPMD</td>
<td>581</td>
<td>25</td>
<td>23.2</td>
<td>NT &amp; Lin.</td>
</tr>
<tr>
<td>ANL macros</td>
<td>146</td>
<td>20</td>
<td>7.3</td>
<td>NT &amp; Lin.</td>
</tr>
<tr>
<td>TreadMarks™ API</td>
<td>326</td>
<td>13</td>
<td>25.1</td>
<td>NT &amp; Lin.</td>
</tr>
<tr>
<td>HLRC API*</td>
<td>137</td>
<td>25</td>
<td>5.5</td>
<td>NT &amp; Lin.</td>
</tr>
<tr>
<td>JiaJia API (subset)*</td>
<td>125</td>
<td>21</td>
<td>5.6</td>
<td>NT &amp; Lin.</td>
</tr>
<tr>
<td>Dist. POSIX threads</td>
<td>725</td>
<td>51</td>
<td>14.2</td>
<td>Linux</td>
</tr>
<tr>
<td>Win32 threads (subs.)</td>
<td>988</td>
<td>42</td>
<td>23.5</td>
<td>NT</td>
</tr>
<tr>
<td>Cray Shmem API</td>
<td>505</td>
<td>29</td>
<td>17.4</td>
<td>Linux</td>
</tr>
</tbody>
</table>

* Based on SMP/SPMD
Completing the picture

Shared Memory application

Shared Memory programming model

Cluster 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
PET Image Reconstruction

PET Scanner at the TU clinic „Rechts der Isar“

Output of the PET scanner

Reconstr. Imagevolume
PET Image Reconstruction / Results

- Cluster: 6 Dual SMPs (450 MHz, Intel Xeon)
- Dataset: Whole body with 282 slices (130 MB)
Summary

- SCI Clusters offers excellent base
  - But: need to know how to exploit it

- Middleware to bridge gap to applications
  - Support of various programming paradigms
  - Low-level implementation to fully utilize SCI

- Shared Memory
  - SISCI = Raw access to SCI
    - Good base for low-level software
  - HAMSTER = Global virtual memory
    - Base for true shared memory programming
Outlook on Lab Session

- Tomorrow afternoon

- Experiments with SISCI
  - Installation of SISCI library
  - Simple code to create and map segments
  - Play with add-ons

- Experiments with HAMSTER
  - Installation
  - Sample programming model SPMD
  - Test code: SOR
  - Play with add-ons
For the curious...

- **SMiLE homepage:**
  
  http://smile.in.tum.de/

- **HAMSTER homepage:**
  
  http://hamster.in.tum.de/

- **Email contact:**
  
  schulzm@in.tum.de