Technische Universität Chemnitz

Sonderforschungsbereich 393

Numerische Simulation auf massiv parallelen Rechnern

Khaled Ragab Wolfgang Rehm

CHEMPI: Efficient MPI for VIA/SCI

Preprint SFB393/01-14

Preprint-Reihe des Chemnitzer SFB 393

SFB393/01-14 **March 2001**

Contents

Author's addresses:

Khaled Ragab
Wolfgang Rehm
TU Chemnitz
Fakultät für Informatik
D-09107 Chemnitz

http://www.tu-chemnitz.de/informatik/RA/

Abstract

Although many ideas about protected user-level network access have been around for a while there still hasn't been a satisfying solution to combine extreme low-latency and high-bandwidth over a wide range of message sizes. Uniting the Scalable Coherent Interface (SCI), a technology to build distributed shared memory as well as message passing systems on top of it, and the Virtual Interface Architecture (VIA), a generic architecture aiming at low latency, seems to be a proper solution to tackle the problem. To reap the benefits of this approach we are developing a new Message Passing Interface (MPI) library, called **CHEMPI**[1]. In this paper we give a brief introduction into some relevant design principles of CHEMPI and prove its potential regarding latency and bandwidth on base of a comparative measurement including two other computing interface technologies.

Keywords: MPI, VIA, SCI, shared memory, and communication protocols.

1 Introduction

MPI (Message Passing Interface) is an established and de-facto standard for data exchange based on the message-passing paradigm for developing portable parallel applications [1][2]. The MPI Forum has introduced the MPI standard in 1996. MPI has a rapidly growing community as a standard user Application Programming Interface (API) for parallel programming. One of the oldest MPI libraries is MPICH, [3]. At present, there are a number of MPI implementations.

The implementation of an MPI library on top of the Scalable Coherent Interface (SCI) can easy be done by emulating message passing on shared memory, but this implies that the CPU participates active in the data transfer. The performance in terms of latency for sending and receiving small message sizes is high, while the system throughput for sending and receiving large message sizes is low due to the active involvement of the CPU for message transfer. The CPU copies the data from local memory into imported SCI memory that consumes expensive CPU cycles with memory copy operations. The DMA engine offers an alternative solution for this problem. For instance Dolphin's PCI-SCI network cards offer a DMA engine. But it can't be used to realize some kinds of protected-user level DMA. To avoid non authorized accesses to memory it is necessary to pass the operating system kernel with the DMA transfer, because only the operating system kernel is able to perform this security check. The DMA engine of the Dolphin PCI-SCI cards D310 showed a lower bandwidth for DMA than SCI shared memory, [8],[9]. To eliminate the operating system kernel from the critical path of the communication operations, three major IT companies[2] introduced the Virtual Interface Architecture (VIA) in December 1997 [5]. The VIA has two communication methods: Send/Receive and RDMA. Both methods are based on a data structure (so-called descriptor) that is used to describe a data movement request. Consequently, the implementation of an MPI library on VIA can not achieve ultra low latencies for small message size as it can be done in SCI by using simple memory references. However the protected user-level DMA engine of VIA offers high bandwidth for

[1]This work is part of the GRANT SFB 393/B6 of the DFG (German national Science Foundation)

[2]Intel, Compaq, and Microsoft.

the MPI library. Hence, the idea to implement a new MPI based on the conjunction of the two network cards looks good and has a great sound.

In the following section we present an overview of some important members of the abundance of MPI libraries that exploit the hardware capabilities of VIA or SCI. In section three we present the requirements for a new MPI library. Following we describe the structure of our new MPI library called **CHEMPI** in section four. In section five we present the implementation of the device dependent layer in CHEMPI. In section six, we prove the send/receive capability of CHEMPI on base of comparative measurements with two other MPI libraries. The last section is related with the future work and conclusion.

2 An overview of relating MPI implementations

MPI/Pro [4], is a product of the MPI software Technology Inc., it was designed and implemented specifically to target Virtual Interface Architecture networks and was designed to take full advantage of VIA. It has a number of features that make it a valuable tool for efficient programming of clusters of workstations. MPI/Pro provides: multi-threaded design, user level thread safety, optimized protocols for short and large messages, optimized persistent mode of MPI point-to-point operations, multidevice architecture, multiple queues for receive requests, and asynchronous method of synchronization and notification.

ScaMPI, [6] is a commercial MPI implementation for SCI connected clusters of workstations. ScaMPI was designed to take the advantage of SCI's shared address space architecture. ScaMPI provides thread safe implementation, fault tolerance, scalability, and high-performance. SCI-MPICH, [7] is a development of MPICH. It has an ADI-2 device (MPID layer) for SCI-adapters that enables MPICH on SCI-connected clusters. The implementation of the SCI-specific ADI-2 device is based on the SMI library that in turn uses the SISCI API and the IRM driver of the Dolphin PCI-SCI adapter.

MPI/Pro did not support a protocol for SCI shared memory so that the ultra-low latencies for short message sizes and synchronization are not reachable. ScaMPI, and SCI-MPICH did not support a protected user-level DMA that is suitable for sending and receiving large message sizes. From this point arose out the strong need for new MPI implementation, that offers the optimal conditions for the conjunction of VIA and SCI to reap the benefits and to avoid the disadvantage of each network card.

3 Requirements for a new MPI

The first design requirements of a new MPI library are achieving **performance** and **portability**. The portability is achieved by separating a hardware dependent functionality in a special layer (so-called device) similar to MPICH, [3]. In numerous cases clusters of the workstations use heterogeneous networks that raise the requirement to implement a MPI library that provides an excellent support for multiple devices. Another general requirement is to support a thread safe programming model to achieve a high degree of parallelism of communication and computation.

Furthermore MPI-2 [2] functionality such as dynamic process creation should be integrated.

Special requirements for supporting SCI shared memory as well as VIA are:

- The registration and deregisteration memory for VIA require a VIA memory management of the data buffers. Memory registration is a high-overhead operation. So, it is recommended to reuse the same registered segment multiple times.
- A shared memory protocol that will be used for short message sizes.
- A protocol that uses the VIA RDMA mechanism for large message sizes.

4 CHEMPI

The CHEMPI (CHEmnitz MPI) project [11] was started in spring 1999 with the objective to fulfill the requirements mentioned above. CHEMPI is divided into two main layers and an interface between these layers as shown in Figure 1:

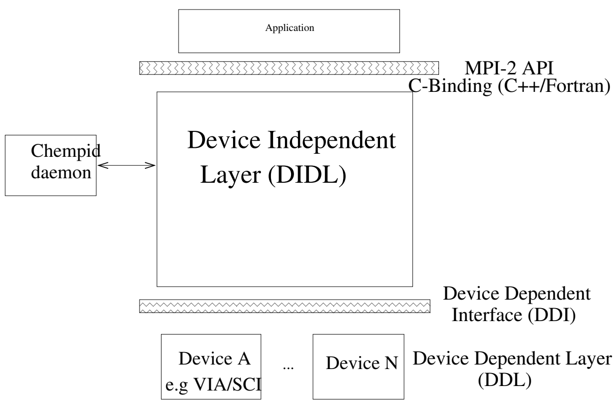

Figure 1: Raw structure of CHEMPI

- **Device Independent Layer (DIDL)** contains a large amount of code and it is divided into the following modules: data-types, MPI point-point and collective operations, connection handling, communicators, groups, topologies, error handling, memory management, process management, and parallel I/O. These modules are still under development. In addition, CHEMPI has a small daemon (chempid) that management the distribution of the network information, and MPI applications. The chempid daemon is still under development for supporting the requirements of MPI-2, such as dynamic process creation.
- **Device Dependent Interface (DDI)** has two parts:
  1. The core functionality has the following parts:
     (a) initialization and shutdown
     (b) basic point to point communication (synchronize/normal and blocking/nonblocking)

(c) send and receive of system messages

2. The optional functionality has the following parts:
   (a) collective operations
   (b) one-sided communication
   (c) other point to point communication modes
   (d) memory management

- **Device Dependent Layer (DDL)**, it is a layer through which the actual communication facilities are accessed. It defines a set of point-point send and receive operations that are required by the upper layer of CHEMPI. In the next sections we will present two devices: SCI device and VIA/SCI device.

The structure of CHEMPI, as shown in Figure 1, provides the ability to generate a new MPI implementation for a new (or not yet supported) communication network with small effort because the effort to implement a DDL part is much smaller than the effort needed to implement the full MPI functionality. Therefore, the structure of CHEMPI provides one of the important characteristics of the software engineering so-called **portability**.

5 CHEMPI Devices

In September 1999 VIA was integrated into the Linux kernel [10] emulating message passing based on the distributed shared memory mode of SCI. The early CHEMPI device [11] was implemented based on this emulator that integrated the VIA and the SCI modes. MPI point-to-point operations implemented on top of this device have been running correctly. But these operations achieved only very low bandwidth and a very high latency. To overcome these limits our group is developing a new VIA/SCI network card [8][9] that hasn't been finished yet. This is why we implemented the CHEMPI device for the Dolphin PCI-SCI adapter D310, and Giganet cLAN[3],[12] with 32/64-bit 33MHZ PCI host adapters cLAN1000. In the next parts we present the implementation of this device.

5.1 The SCI device

The implementation of the SCI device is based on the SISCI API and the Dolphin SCI adapter D310. The implementation of this device has three parts:

- **Initialization:** During the initialization of the processes which form the MPI application, the SISCI API and the IRM[4] driver of the Dolphin PCI-SCI adapter are required to establish shared SCI memory segments mapped into the process's address space. The initialization process starts by creating the local segments[5] for all the other processes. Each segment

[3]The cLAN for Linux product family is a hardware implementation of the VIA standard.

[4]Interconnect Resource Manager,[13]

[5]Local segment is a memory that segment located on the same processor where the application runs and accessed using the host memory interface.

has a unique segment identifier. Afterwards the initialization process makes sure that all of the local segments are accessible by an SCI adapter, maps these local segments that were created before into the addressable space of the device implementation, and makes all of these local segments visible to the remote nodes. Following it connects these segments to the associated remote segments[6], and maps these remote segments into the addressable space of the device implementation. In addition, the initialization processes should create a so called sequence for all the other processes. This sequence is used to check if errors have occurred during a data transfer. An example of connections established after the initialization for four nodes is shown in Figure 2.

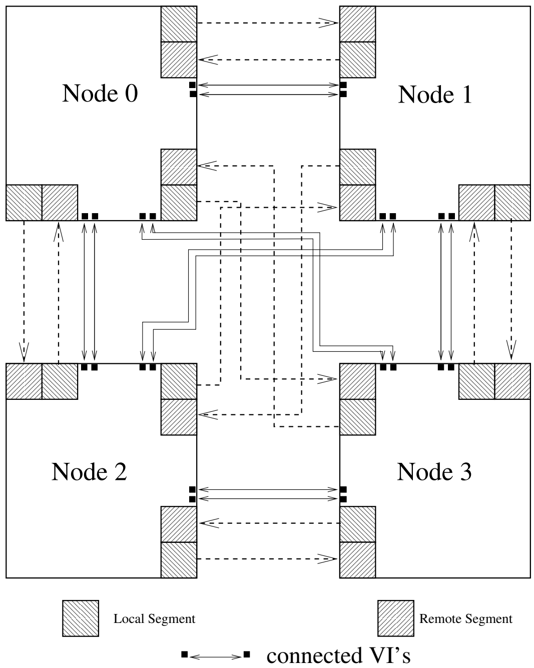

Figure 2: VIA/SCI device after initialization

- **Communication:** The SCI device handles point to point blocking communication by direct accesses to the user address space of the process and supports only a message transfer protocol for small message sizes that will be used in SCI/VIA device. This protocol is implemented as follows:

  1. **At the sender side**, the data transfer from the local segment to the remote segment is done by copying the data in the user address space into the remote map address of the destination and post a ready flag. This flag is used for synchronization. Then, the sender should wait until the data has arrived at the receiver. Using another flag called (Completed) that will be posted by the receiver can do this.

[6]Remote segment represents a segment residing on a remote node and accessed via SCI link

2. **At the receiver side**, it waits until the data has arrived and the ready flag is posted. After the receiver has copied the data from its local segment into its application buffer, it should post the Completed flag.

The protocol for point to point communication uses SCI remote writes accesses instead of remote read access to get a high bandwidth and low latency.

- **Shutdown:** The shutdown process should do the following: disconnect the remote segments, hide the available local segments to the remote nodes, unmapped the local segments, remove the local segments, and remove the sequences.

5.2 The VIA/SCI device

The implementation of the VIA/SCI device is based on the SISCI API, the Dolphin SCI adapter D310 and the Giganet cLAN Software version 1.0.3 for linux operating system and the cLAN Cluster Switch with cLAN Host Adapters. A VIA/SCI contains of three parts:

- **Initialization:** In addition to the initialization process we mentioned before in section 5.1, the initialization process does the following:
  1. Opens the network interface controller and creates a protection tag,
  2. Creates an instance of a virtual interface (VI) for each process,
  3. Makes the connection between the local VI end point and the other VI's based on the client/server model (i.e., Each two VI's are connected between each couple of MPI tasks after the initialization). An example of connections established after the initialization for four nodes is shown in Figure 2.
  4. Allocates and registers memory for descriptors, and for user data buffers. Memory registration is a high-overhead operation. CHEMPI reduces these overheads by registering memory for descriptors outside send/receive operations, and reusing the same registered segments multiple times.

- **Communication:** The VIA/SCI device handles point to point communication in two different ways depending on the message size. It uses the SCI Send/Receive protocol for small message sizes[7], and RDMA Write[8] protocol for large messages. The VIA/SCI device implement these protocols as follows:
  1. **Small message protocol:** This protocol uses the same protocol we mentioned before in the SCI device at the sender and the receiver,
  2. **Large message protocol:** This protocol is based on the VIA descriptors for transferring data as well as on SCI for posting the control variables (flags) that are used for synchronization between the sender and the receiver processes. We use a protocol called "**Zero Copy VIA protocol**". In the next section, we will discuss this protocol.

[7]The small message is ranged from 0.. 2048 bytes.
[8]VIA RDMA Write for message size large than 2048.

- **Shutdown** In addition to the shutdown process we mentioned before in the section 5.1, the shutdown process should do the following:
  1. Free and deregister the memory used for the descriptors,
  2. Terminate the connection between the local VI end point and the other VI's, and destroy the local VI.
  3. Destroy the protection tag, and shutdown the name service.
  4. Remove the association between the local process and the VI NIC by closing the NIC.

5.3 Zero Copy VIA protocol.

The word Zero Copy means that the data transfer requires no additional copy operations neither at the sender nor at the receiver. This protocol should start by making sure that the user buffers at the sender and the receiver is registered. It requires a descriptor only at the sender. The scenario of the this protocol at the sender side and the receiver side, as shown in figure 3, is:

- **At the sender side**,
  1. The sender checks if the user buffer is already registered in the VIA memory management and register it if not.
  2. It fills out the message envelope (tag, source, message size, and communicator) in the remote segment.
  3. The sender must know the user address space and the memory handle on the receiver side. Now, it should wait until the receiver posts a ring, and this information is written to the sender local segment.
  4. The sender splits the message of size M[9] into N chunks, and builds N descriptors and then sends them.
  5. The sender must wait until all these descriptors have been completed.
  6. The sender should post a flag (so called completed flag) to the receiver when all descriptors have been completed. This flag means the message of size M has been sent completely.
  7. It sets the user buffer in the VIA memory management unused.
- **At the receiver side**,
  1. The receiver checks if the user buffer is already registered in the VIA memory management and register it if not.
  2. It waits until it finds a match for the sender message envelope.

[9] In case of M is large than the descriptor length field (65535 bytes).

3. When the receiver finds a match for the sender message envelope, it posts a ring to the sender and fills out the remote segment by its user address space and memory handle.

4. It waits until the completed flag is posted by the sender to make sure that the message has arrived at the receiver.

5. It sets the user buffer in the VIA memory management unused.

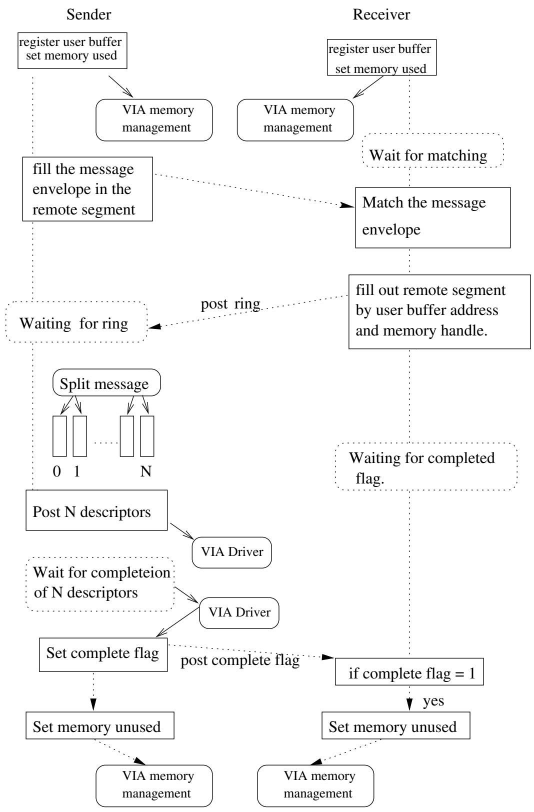

Figure 3: Zero copy protocol

This protocol was implemented based on the **VIA RDMA Write**. The sender initiates the RDMA write process to transfer the data. The major reason of using the RDMA write instead of the RDMA read is that the Giganet VIA implementation supports only the RDMA write. As mentioned above, the sender needs to know the user address and the memory handle at the receiver side. CHEMPI achieves this requirement by using the SCI shared memory for transferring this information from the sender side to the receiver side while MPI/Pro [4] achieves that by an additional VIA send/receive operations. The VIA send/receive protocol can not achieve the low latency as it can be done in SCI.

6 Performance

Our test environment consisted of two Pentium III 450 MHz machines. They were equipped with 128MB and were connected by an SCI ring topology based on Dolphin's D310 PCI-SCI bridges. In addition, two machines had cLAN 1000 VIA adapter from Giganet Inc. Our tests machines were running RedHat Linux 6.0. For ScaMPI tests we used Scali's SPP version 2.0, while the measurements on GigaNet were done using MPI/Pro by MPI software Technology Inc., which is the only MPI with support for Giganet's cLAN hardware. For the measurements of latency and bandwidth, we use a simple ping-pong benchmark between two MPI processes. Each process executes blocking send and receive operations to wait for an incoming message (MPI_Recv()) and immediately responds (MPI_Send()) once it has arrived. The resulting round-trip times are then halved to give the effective latency, from that we derive the bandwidth.

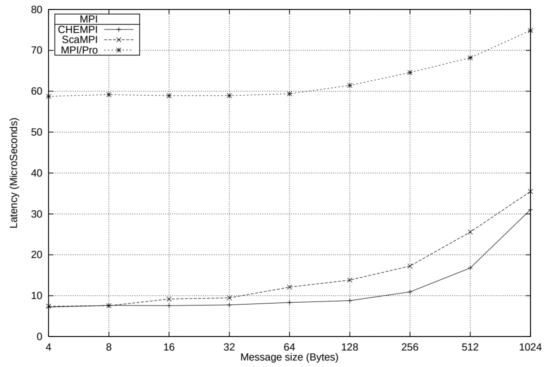

Figure 4: Ping-pong latency half round-trip between 2 MPI processes with message sizes 0Bytes .. 1024Bytes

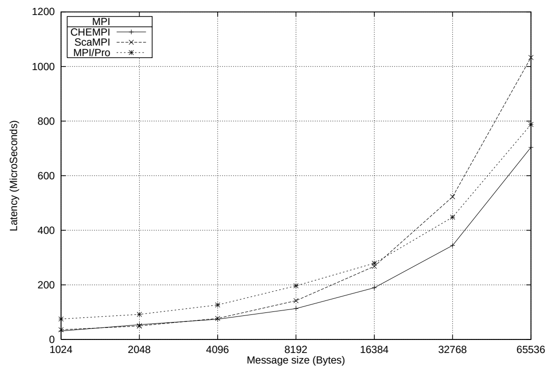

Figure 5: Ping-pong latency (half round-trip) between 2 MPI processes with message sizes 1KB .. 64KB

The graphs are shown in figures (4,5,6) present the latency (half round-trip) measurements between two MPI processes. Figure 4 shows the MPI blocking send/receive latency of three MPI

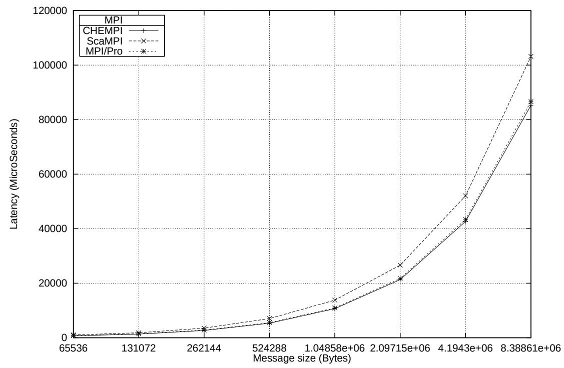

Figure 6: Ping-pong latency (half round-trip) between 2 MPI processes with message sizes 64KB .. 8MB

libraries (CHEMPI, MPI/Pro, ScaMPI), with message sizes between zero bytes and 1024 bytes. It can be seen clearly in this figure that CHEMPI has the smallest latency as compared to ScaMPI and MPI/Pro. This figure shows that MPI/Pro needs more than twice as much time as CHEMPI needs for sending/receiving a message of 1024 bytes. This is a result of VIA communications that are based completely on explicit descriptor processing and have no way to achieves low latency as it can be done in SCI by using simple remote memory references. Figure 5 shows MPI blocking send/receive latency of three MPI libraries (CHEMPI, MPI/Pro, and ScaMPI), with message sizes between 1KB and 64KB. This figure shows that ScaMPI gives better results than MPI/Pro up to the crosspoint at 16KB. After this point MPI/Pro is faster. This confirms the measurements presented in r14. This is from memory copy overhead in ScaMPI for long messages. Figure 6 shows MPI blocking send/receive latency of three MPI libraries (CHEMPI, MPI/Pro, and ScaMPI), with message sizes between 64KB and 8MB. It shows that CHEMPI has the smallest latency, although MPI/Pro has only a slight difference to CHEMPI. This comes from the fact that CHEMPI uses SCI for the required synchronization. Figure 7 shows the MPI blocking send/receive bandwidth of all the MPI libraries. It shows that CHEMPI and ScaMPI achieve higher bandwidth than MPI/Pro for small messages, while CHEMPI and MPI/Pro provide better performance compared with ScaMPI for large messages. The reasons were explained above. As seen in all figures, CHEMPI has the best performance (high bandwidth and low latency). This is a result of CHEMPI has a dynamic adaptable device VIA/SCI. The CHEMPI VIA/SCI device use different protocols based upon the message size and exploit the advanced hardware capabilities of the two hardware network cards (Dolphin's PCI-SCI adapter D310, and GigaNet cLAN adapter).

7 Conclusion and future work

The implementation of the CHEMPI validates the idea of combining VIA and SCI by achieving high bandwidth and low latency compared with other MPI libraries based upon only one of these principles. Hence, we can imagine what performance gains we will get when we have the

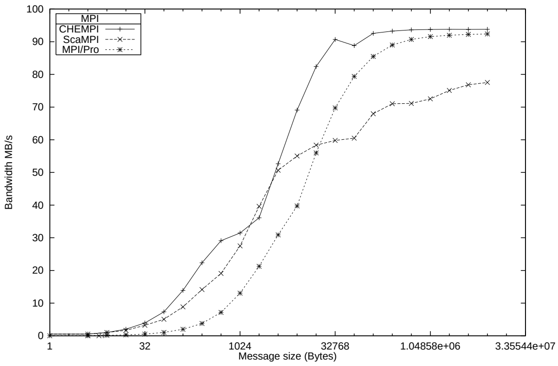

Figure 7: Ping-pong bandwidth (half round-trip) between 2 MPI processes

two concepts VIA and SCI integrated in the hardware of a unique network card. Currently our research group is developing an appropriate network card [9]. Although CHEMPI is still under development our first measures have already shown that CHEMPI is a promising MPI library. Our future work is concerned with the implementation of further send/receive modes such as nonblocking and collective operations as well. Furthermore we aim at integrating of advanced MPI-2 functionalities.

References

[1] Message Passing Interface Forum. MPI: A message–passing interface standard Vers. 1.1. http://www.mcs.anl.gov/mpi/standard.html, June 1995.

[2] The MPI Forum. The MPI message-passing interface standard Vers. 2.0, May 1998.

[3] W. Gropp, E. Lusk, N. Doss, A. Skjellum. Portable Implementation of the MPI message–passing standard. Argone National lab. and Mississippi state university. 1996.

[4] Rossen Dimitrov and Anthony Skjellum. An Effizient MPI Implementation for Virtual Interface (VI) Architecture–Enabled Cluster Computing. In *Proceedings of the Third MPI Developers Conference*, March 1999.

[5] Intel, Compaq, and Microsoft. Virtual Interface Architecture Specification V1.0. http://www.viarch.org.

[6] L.P. Huse, K. Omang, H. Bugge, H. Ry, A.T. Haugsdal, and E. Rustad ScaMPI – Design and Implementation. Scali A, Norway. http://www.scali.com/

[7] J. Worringen, T. Bemmerl MPICH for SCI–Connected Clusters Proc. SCI Europe'99, pp. 3-11 Toulouse, France, Sept. 1999.

[8] M. Trams, W. Rehm, and F. Seifert, An advanced PCI-SCI bridge with VIA support, Workshop Cluster Computing, Univ. Karlsruhe, Faculity of computer science, Mar. 1999.

[9] Mario Trams and Wolfgang Rehm. A new generic and reconfigurable PCI-SCI bridge. Proceedings of SCI Europe'99, Toulouse, September, 1999.

[10] Friedrich Seifert Development of system software to integrate the virtual interface architecture (VIA) into the Linux operating system kernel for optimized message passing. Diploma Thesis, Dept. of Computer Science, University of Technology, Chemnitz, 1999.

[11] S. Schindler, W. Rehm, C. Dinkelmann. An optimized MPI library for VIA/SCI cards, Asia-Pacific International Symposium on Cluster Computing (APSCC'2000).

[12] http://www.giganet.com/products/index.htm

[13] Dolphin Interconnect Solutions Inc., http://www.dolphinics.no/

[14] Fredrich Seifert, Daniek Blakanski, and Wolfgang Rehm. Comparing MPI performance of SCI and VIA, Proceeding of SCI Europe'2000, Munich, Germany, August, 2000.

Other titles in the SFB393 series:

01-01 G. Kunert. Robust local problem error estimation for a singularly perturbed problem on anisotropic finite element meshes. January 2001.

01-02 G. Kunert. A note on the energy norm for a singularly perturbed model problem. January 2001.

01-03 U.-J. Görke, A. Bucher, R. Kreißig. Ein Beitrag zur Materialparameteridentifikation bei finiten elastisch-plastischen Verzerrungen durch Analyse inhomogener Verschiebungsfelder mit Hilfe der FEM. Februar 2001.

01-04 R. A. Römer. Percolation, Renormalization and the Quantum-Hall Transition. February 2001.

01-05 A. Eilmes, R. A. Römer, C. Schuster, M. Schreiber. Two and more interacting particles at a metal-insulator transition. February 2001.

01-06 D. Michael. Kontinuumstheoretische Grundlagen und algorithmische Behandlung von ausgewählten Problemen der assoziierten Fließtheorie. März 2001.

01-07 S. Beuchler. A preconditioner for solving the inner problem of the p-version of the FEM, Part II - algebraic multi-grid proof. March 2001.

01-08 S. Beuchler, A. Meyer. SPC-PM3AdH v 1.0 - Programmer's Manual. March 2001.

01-09 D. Michael, M. Springmann. Zur numerischen Simulation des Versagens duktiler metallischer Werkstoffe (Algorithmische Behandlung und Vergleichsrechnungen). März 2001.

01-10 B. Heinrich, S. Nicaise. Nitsche mortar finite element method for transmission problems with singularities. March 2001.

01-11 T. Apel, S. Grosman, P. K. Jimack, A. Meyer. A New Methodology for Anisotropic Mesh Refinement Based Upon Error Gradients. March 2001.

The complete list of current and former preprints is available via
`http://www.tu-chemnitz.de/sfb393/preprints.html`.