Enabling a PC Cluster for High-Performance Computing[1]

Hermann Hellwagner, Wolfgang Karl and Markus Leberecht

Lehreinheit für Rechnertechnik und Rechnerorganisation / Parallelrechnerarchitektur (LRR-TUM)
Institut für Informatik
Technische Universität München
Munich, Germany

Abstract

Due to their excellent cost/performance ratio, clusters of PCs can be attractive high-performance computing (HPC) platforms. Yet, their limited communication performance over standard LANs is still prohibitive for parallel applications. The project “Shared Memory in a LAN-like Environment“ (SMiLE) at LRR-TUM adopts Scalable Coherent Interface (SCI) interconnect technology to build, and provide software for, a PC cluster which, with hardware-based distributed shared memory (DSM) and high-performance communication characteristics, is regarded as well suited for HPC.

The paper describes the key features of the enabling technology, SCI. It then discusses the developments and important results of the SMiLE project so far: the development and initial performance of a PCI/SCI interface card, and the design and initial performance results of low-latency communication layers, Active Messages and a sockets emulation library.

1 Introduction

In recent years, networks of workstations (NOWs) have become widely adopted for parallel, high-performance computing (HPC), particularly in academia and small or medium enterprises. The advantages of NOWs are manifold: leading-edge technology, high performance, and high dependability of the compute nodes (off-the-shelf RISC workstations); their low utilization; mature connectivity, both in hardware and software; and the availability of software systems like Parallel Virtual Machine (PVM) [8] that allow to harness the distributed compute power for parallel applications at practically no costs.

One approach to proliferate HPC capabilities even further is to adopt networked PCs for parallel work. PCs are ubiquitous and have excellent cost/performance ratios; also, today’s PC operating systems like Linux and Windows NT provide reliable communication support over standard LANs. With PVM being available for Linux and, more recently, for the Win32 programming interface [19], clusters of PCs are becoming attractive platforms for parallel computing.

However, w.r.t. parallel processing, both NOWs and PC clusters usually suffer from a severe drawback: poor communication performance. This stems from the fact that communication mechanisms over standard networks, e.g., PVM message-passing, are based on the TCP/IP suite of protocols. These are operating system-level, heavy-weight protocols developed for transferring data reliably over unreliable, probably widely distributed networks. The resulting communication architecture and very large processing overheads (analyzed in detail e.g., in [5, 7]) limit the attainable communication bandwidth and cause high end-to-end message latencies, usually in the *milliseconds* range. This is two to three orders of magnitude inferior to state-of-the-art multiprocessors [7, 10] and makes conventional clusters ill-suited for fine-grained, communication-intensive parallel applications.

It is interesting to note that, due to the software overheads, this drawback also holds for so-called high-

1. Appeared in: *The SPEEDUP Journal*, Vol. 11, No. 1, June 1997.

speed networks, e.g. those based on ATM. As a point of reference, communication over BSD TCP/IP sockets on an ATM-based workstation cluster is reported to incur a round-trip latency of 3.9 ms and to achieve a maximum data transfer rate of only 2 Mbytes/s (of 12.5 Mbytes/s raw network bandwidth) [7].

Therefore, substantial research into more efficient communication mechanisms for workstation clusters has been conducted in recent years. The bottom line of this research is to more closely and reliably couple the nodes in a cluster environment through dedicated interconnect and network interface hardware, and to tailor data transfers and control to the needs of parallel computing (in particular, low latency), by removing the operating system and, as far as possible, software processing in general, from the critical communication path. State-of-the-art cluster communication architectures achieve latencies as low as a few (tens of) *microseconds* and data transfer rates of tens of Mbytes/s. The interested reader is referred to the proceedings of recent workshops on this topic, e.g. [3, 12, 13, 17].

The SMiLE project at LRR-TUM is one such project to establish a compute cluster with communication characteristics that make it well-suited for HPC. The project adopts and implements SCI interconnect technology, most notably the hardware DSM specified in the SCI standard, to closely couple conventional Pentium or Pentium Pro-based PCs. Various software systems for parallel computing are being adapted to make efficient use of this communication architecture.

This paper describes the key features of the enabling technology, SCI, and outlines the objectives of the SMiLE project (section 2). In section 3, the design and initial performance figures of a PCI/SCI adapter card are presented. Section 4 describes the design and initial performance results (on an SCI-based workstation cluster) of low-latency communication layers, namely an Active Messages library and a user-level libary emulating BSD sockets. Conclusions and an outline of further work are given in section 5. The focus in this paper is on the DSM and how it is exploited in software to achieve high-performance communication.

2 SCI and the SMiLE Project

2.1 Key Features of SCI

SCI is a recent standard [20] that specifies innovative interconnect hardware and protocols to tightly connect up to 64k nodes (e.g., multiprocessors, workstations or PCs, bus bridges, network interfaces, I/O adapters) into a high-speed network [9, 10]. Having emerged from an attempt to define a “superbus“ standard, SCI defines bus-like services, but offers fully distributed solutions for their realization. The most notable of these services are a single physical 64-bit address space across SCI nodes and related transactions for reading, writing, and locking DSM areas. This hardware DSM spans up to 2^{16} nodes, with up to 2^{48} addresses per node. Complex, distributed cache coherence protocols for the DSM have been developed so that SCI systems with NUMA as well as CC-NUMA characteristics can be built. Transactions for efficient messaging as well as for fast event notification or synchronization are also specified.

SCI avoids the physical limitations of computer busses by employing unidirectional point-to-point links only. Thus, there are no principal impediments to scalability; the links can be made fast, their performance can scale with improvements in the underlying technology, and they can be implemented with serial or parallel transmission lines, over various distances (up to kilometers), and based on various media (copper or fiber optic). At the physical layer, SCI initially specifies link bandwidths of 1 Gbit/s using serial links and 1 Gbyte/s using parallel links over small distances.

At the logical layer, SCI defines packet-based, split-transaction protocols. Up to 64 transactions can be outstanding per node, allowing for each node to pipeline multiple packets into the network. The interconnect bandwidth can thus be utilized efficiently, and the latencies of e.g., DSM accesses can be hidden effectively for the node.

The basic building blocks of SCI networks are small rings (ringlets). Large systems are built of rings of rings, rings interconnected via SCI switches, or multistage interconnection networks that can be constructed from ringlets. Further, a standardized interface of a node to the SCI network is proposed, with an incoming and an outgoing SCI link per node. The protocols defined for this basic model provide for fair bandwidth allocation and network administration; they are deadlock-free and robust. Maintenance of data integrity and error detection is primarily done in hardware, for instance by hardware checksum computation and by time-out logic that supervises outstanding transactions and performs retries in case of errors.

The major benefit of SCI networks and protocols is that communication can be performed at user level via the hardware DSM, by load and store operations into memory regions mapped from remote memories. This translates into latencies in the low *microseconds* range even in a cluster environment, e.g., [18]. For high-volume communications, the SCI messaging transactions can be employed which use DMA and SCI hardware protocols for data transfers. Throughput close to the bandwidth provided by the network interface can be achieved in this manner, e.g., [18].

2.2 The SMiLE Project

The SMiLE project at LRR-TUM has the objectives to enable a low-cost cluster built from conventional PCs for high performance computing, by implementing a cluster-wide DSM with NUMA characteristics in hardware (outlined in Figure 1); to adapt parallel programming models and application programs to this platform (cf. Figure 2); and to use it as a vehicle for research into the efficient use of parallel systems with DSM.

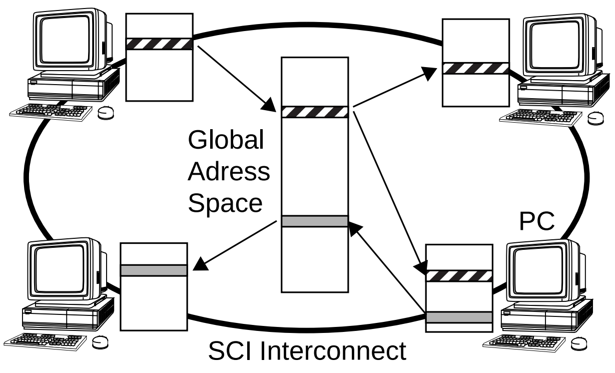

Figure 1. SMiLE SCI Cluster Model

According to these objectives, the work within the project falls into three categories.

Hardware developments to set up a PC cluster with DSM using SCI

The main component, a PCI/SCI interface card, is presented in section 3. Additional hardware is currently being developed that allows to monitor the node’s outgoing SCI transactions, primarily remote memory references. This information will be used to analyze and optimize the performance of DSM programs.

Adaptation and optimization of parallel APIs for the SCI cluster

In addition to SCI device drivers for Linux and Windows NT, parallel programming models (APIs) are being made available on the SCI cluster. The goal here is to exploit the specific features of the SCI interconnect and the DSM so that applications using these APIs can largely benefit from high performance communication. Work so far (performed on Sun workstation testbeds using commercial SBus/SCI adapter cards) comprises adaptation of PVM direct data transfers [11] and of a Unix SVM system (Shared Virtual Memory), and the development of user-level libraries for communication via Active Messages and emulated BSD sockets. The latter two libraries are described in section 4. Work on implementing a low-level SCI API that is currently under standardization [21], and on adapting and optimizing full PVM as well as a distributed, DSM-based threads package (Figure 2), is about to commence.

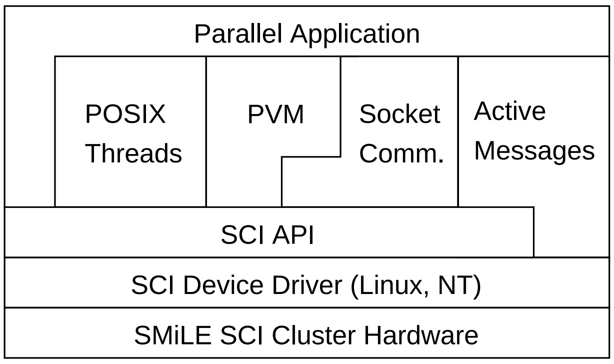

Figure 2. SMiLE SCI Cluster Software Layers

Development and test of concepts, programming models, and tools for the efficient use of a DSM cluster

In SCI DSM, remote memory accesses are performed in hardware; yet, they are about an order of magnitude more expensive than local accesses (NUMA) [15]. Therefore, it is of crucial importance for the efficiency of parallel programs to perform local memory references as far as possible. It is the long-term goal of our research to develop, and experiment with, program annotations, run time statistics and automatic run time tools, and interactive performance analysis tools, to establish, exploit, or analyze data locality. An important component of this research is the hardware monitor mentioned above.

3 The SMiLE PCI/SCI Adapter Card

Since a commercial PCI/SCI interface card was not available at the start of the SMiLE project and since we wanted to be able to integrate monitoring hardware into the SCI interface at a later stage, we developed our own PCI/SCI adapter card. It is described in detail in [1, 2]. The adapter serves as the interface between a PC's PCI bus and the SCI interconnect. It intercepts PCI transactions, e.g. memory accesses to certain addresses, translates them to SCI transactions, and forwards them into the network; vice versa, it takes packets destined for the node off from the incoming link, translates them into PCI transactions, and forwards them to the PCI bus. The PCI/SCI interface is responsible for request/response packet generation and protocol processing.

In addition, the PCI/SCI bridge performs the required address translations. The bridge has an address window of 64 Mbytes in the PCI address space, 32 Mbytes of which can be used to map SCI addresses into (at page granularity). An SCI address comprises an 8-bit node address and a 32-bit node-local address. Thus, physical memory regions on different nodes (in the SCI address space) can be mapped into a node's PCI (i.e., physical) address space, and – using `mmap()` for instance – further on into the processes' virtual address spaces. Using this two-stage address translation scheme, remote memory can be accessed by simple load and store operations, transparent to software. It is important to note that remote data are not cached locally; hence, problems associated with cache coherence and the need to implement the SCI coherence protocols do not arise.

The PCI/SCI interface supports one outstanding read transaction (`readsb`), 16 outstanding write transactions (`writesb`), 16 outstanding messaging transactions (`nwrite64`) that use DMAs for data transfers on the nodes, and one read-modify-write memory transaction (`locksb`) to be used for synchronization primitives. Each outstanding transaction is supervised by a separate timer. Other transactions can be explicitly constructed by software. Up to 64 incoming SCI packets can be buffered in an interface card.

Notice that 16 outstanding writes as opposed to one outstanding read are supported. This was a deliberate design decision based on an observation we made on one of the SCI testbeds: SCI remote reads take about 5.5 µs, whereas remote writes take 0.4 µs only [15]. This is due to the fact that remote reads have to wait for data to arrive, whereas remote writes are buffered in the local SCI interface and immediately acknowledged to the processor. As a consequence of this feature, the SCI software is being designed to exploit remote writes as far as possible.

Figure 3 shows a high-level block diagram of the PCI/SCI bridge. The PCI Unit interfaces to the local PCI bus, performs PCI-to-SCI address translations, and translates PCI transactions into SCI transactions, and vice versa. SCI transactions are buffered in the Dual-Ported RAM (DPR). Control information is passed on to and received from the SCI Unit via a Handshake Bus. For performance reasons, the DPR contains pre-fabricated SCI packets where only address information and data have to be filled in. The SCI Unit is responsible for SCI protocol processing and delivering SCI packets to the node via the DPR. This unit is based on a commercial SCI interface chip, the Link Controller LC-1 from Dolphin Interconnect Solutions. Both major units contain large FPGAs that implement most of the functionality of the SCI interface card.

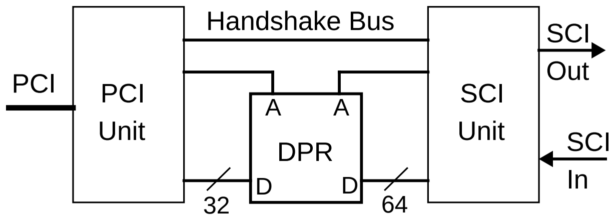

Figure 3. PCI/SCI Adapter Architecture

A prototype of the PCI/SCI bridge has been tested and is currently being optimized. Using the bridge in loopback mode at 18 MHz, “remote“ write latencies of 3.1 µs have been measured. With 25 MHz, this should reduce to about 2.4 µs. Figure 4 shows the results of initial throughput measurements at 12 MHz. With 25 MHz, the throughput (per node) should increase to about 20 Mbytes/s. At block sizes of more than 1024 bytes, the number of outstanding messaging transactions (16 64-byte packets) becomes the limiting factor.

A small number of adapters is currently being manufactured so that the SMiLE cluster can be built in the near future.

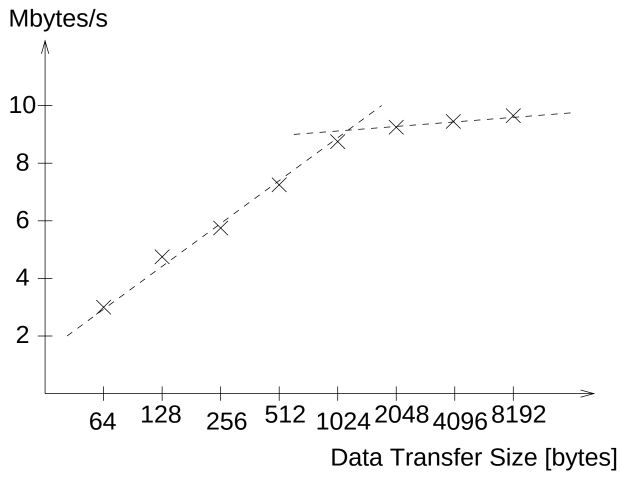

Figure 4. Throughput of DMA-based Data Transfers over SCI (Loopback Mode, 12 MHz)

4 Low-Latency Communication Layers

In order to demonstrate the high-performance communication capabilities of an SCI system, we decided initially to adapt an efficient, user-level communication mechanism, *Active Messages* [6], to an SCI DSM. The second communication system developed emulates at user level a legacy protocol in distributed systems, the BSD sockets interface [16] to the TCP/IP family of protocols. The benefit of this API is that many message-passing libraries in NOWs, e.g., PVM, are built on top of it. Through the sockets library therefore, other programming models are supported on SCI systems as well.

The software developments reported here are currently taking place on an SCI testbed of two SPARC-stations-2, coupled to the SCI network via the SBus-1 adapter card from Dolphin. The software is written in a modular manner so that porting to the SMiLE cluster (under Linux) should pose no problems. Performance tests are also conducted on an SCI cluster of eight UltraSPARC workstations at the University of California, Santa Barbara (UCSB), the nodes of which are equipped with higher-performance Dolphin SBus-2 adapters [14].

4.1 Active Messages

Active Messages (AM) are a light-weight messaging mechanism that allows to reduce the overheads of sending and receiving messages (which directly translates to latency) by at least one order of magnitude, compared to more general mechanisms that provide for asynchronous messaging and mutual protection of concurrent processes, for instance, and that perform multiple data copying and buffering. This is achieved through a different understanding of a message which not only contains the relevant data but also a function pointer identifying a so-called message handler function. Upon receipt of a message, this function is called with the transported data as its parameter, in order to remove the message from the communication layer and deposit it directly in the application’s address space. The message handler is a user-level, non-blocking function, making this concept best suited for short messages. Through this technique, buffering and latencies are reduced to a minimum. Architectures that permit user-level access to the network interface (such as an SCI DSM) benefit the most from the AM concept because context switches into the operating system are avoided. A drawback of this original scheme (AM 1.1) is that there are no protection mechanisms and that programs using AM must execute in an SPMD style.

The SMiLE AM implementation [4] is based on the second version of the AM interface (AM 2.0). This version introduces the abstraction of communication *endpoints* that have to be acquired from the AM layer and that are identified by endpoint tags. These tags can be checked upon message receipt, thus establishing a basic protection mechanism.

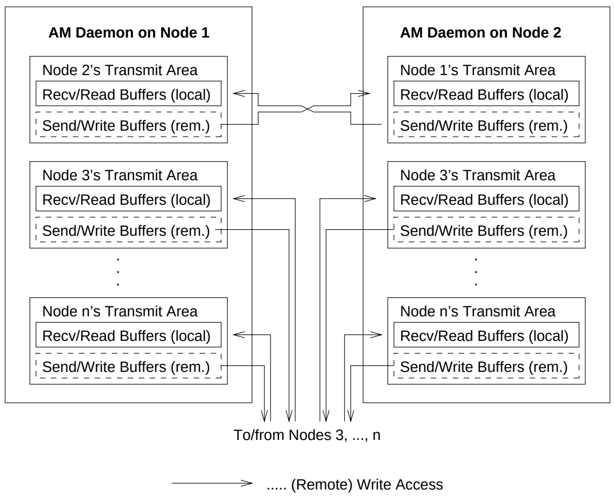
AM Daemon on Node 1
Node 2’s Transmit Area
Recv/Read Buffers (local)
Send/Write Buffers (rem.)
Node 3’s Transmit Area
Recv/Read Buffers (local)
Send/Write Buffers (rem.)
Node n’s Transmit Area
Recv/Read Buffers (local)
Send/Write Buffers (rem.)
AM Daemon on Node 2
Node 1’s Transmit Area
Recv/Read Buffers (local)
Send/Write Buffers (rem.)
Node 3’s Transmit Area
Recv/Read Buffers (local)
Send/Write Buffers (rem.)
Node n’s Transmit Area
Recv/Read Buffers (local)
Send/Write Buffers (rem.)
To/from Nodes 3, ..., n
..... (Remote) Write Access

Figure 5. Structure of the SMiLE AM System

The SMiLE AM system consists of two main components: the AM *daemons*, one per node, that initialize SCI shared memory segments and make them available to other AM daemons; and the AM *library* that implements the AM API and performs the data transfers. As depicted in Figure 5, send and receive pools of the communication endpoints are allocated in the shared SCI address space, with receive buffers being allocated locally and mapped remotely as send buffers. As motivated above, data are transferred via efficient, buffered writes in the SCI DSM only.

Performance measurements on the UCSB SCI cluster show competitive performance behavior of the SCI AM system (Table 1). Our own implementation, depicted in the first row of Table 1, adds little overhead to the raw latency of 9.3 µs, and achieves more than 80% of the network's raw bandwidth. For comparison, performance figures from [14] are given in the next three rows.

Platform / AM Version	Roundtrip Latency [µs]	Throughput (AM Store) [Mbytes/s]
UCSB SCI NOW / AM 2.0 (LRR-TUM)	15	21
UCSB SCI NOW / AM 1.1 (UCSB) *	16.5	13
Myrinet NOW	28.9	34
Meiko CS-2	23	32

* Not optimized

Table 1. Performance Characteristics of Different AM Implementations

4.2 SCI Sockets Library

While common sockets implementations over standard LANs use the regular TCP/IP protocol stack that is usually located in kernel space, a major design goal for the SMiLE sockets implementation has been to achieve data transfers on user level. In the resulting system, *SCI Sockets Library (SSLib)* [22], this is again accomplished through user-mode shared memory accesses, preferably buffered writes. While all socket API calls are implemented (including *out-of-band* data transfers), SSLib is evidently restricted in the sense that connections can only be set up locally within an SCI system.

SSLib's internal architecture resembles somewhat the structure of the AM system. Daemon processes, one per node, are responsible for initiating and maintaining connections and also handle the subtleties pertaining to socket inheritance of child processes. Data transport is performed through remote memory buffers (DSM segments) mutually mapped by the sender and receiver. The two DSM segments required for a (bidirectional) socket are organized as ring buffers, with start and end pointers being located in the DSM and being incremented by buffered writes as well. This facilitates a fundamental form of flow control: a sender blocks on encountering a full buffer, a receiver on an empty buffer. The original design and implementation of the BSD sockets interface requires that a copy operation be performed that transfers the data from a buffer in the sender's address space to the actual send buffer (in SSLib, in the DSM; originally, with TCP/IP, in kernel space), and vice versa at the receiver's side.

Similar to AM, SSLib is written in a hardware independent manner such that porting it to the SMiLE cluster should be without problems. SSLib is built on top of a low-level communication layer that hides the specific implementation of the underlying shared memory (and other communication mechanisms). Thus, the SSLib could readily be tested both on an SCI DSM and a shared-memory multiprocessor. In addition, this layer selects the most efficient communication mechanism for a given data transfer (shared memory operations or message passing using DMA transfers), depending on the data volume to be sent.

Common Unix tools that use sockets extensively, e.g. `ftp`, `ftpd`, `inetd`, have been used to verify the correct behavior of SSLib. Preliminary performance results on the UCSB SCI cluster are encouraging: roundtrip latencies of about 17 µs for a 1-byte packet have been observed.

5 Summary and Further Work

Based on results from the SMiLE project at LRR-TUM, we have shown that clusters of workstations or PCs can be made into attractive HPC platforms, not only in terms of low costs and high computation performance, but also with respect to high performance communication. The SCI interconnect standard is an enabling technology for this endeavor, providing hardware-supported DSM and user-level access to the network. Most notably, this translates into low-latency communication (in the low microsecond range) which is particularly important for parallel applications. Also, high data transfer rates can be attained, close to the raw bandwidth of the network (or network interface). This has been demonstrated by the two communication libraries described in this paper, Active Messages and a socket emulation library. These communication layers will be ported to the SMiLE cluster which will be based on the PCI/SCI adapter card presented here as well.

In this paper, the focus has been on message-based communication. It is our belief that similar benefits can be realized for shared variable-based programming models. As a next step, a distributed thread package making use of the SCI DSM will therefore be developed. This will be supported by a performance analysis tool allowing to monitor and optimize the memory reference behavior of parallel programs in the NUMA SCI system.

Acknowledgements

We would like to thank our students Georg Acher, Michael Eberl, and Josef Weidendorfer for their enthusiasm and their great work in the SMiLE project. The support from Klaus Eric Schauser and his research group at the University of California at Santa Barbara by providing access to their SCI cluster, is also gratefully acknowedged.

References

[1] G. Acher. *Entwicklung eines SCI-Knotens zur Kopplung von PCI-basierten Arbeitsplatzrechnern mit Hilfe von VHDL.* Diplomarbeit, Institut für Informatik, Technische Universität München, Okt. 1996.

[2] G. Acher, H. Hellwagner, W. Karl, M. Leberecht. “A PCI-SCI Bridge for Building a PC Cluster with Distributed Shared Memory”. *Proc. 6th SCIzzL Workshop on SCI-based Local Area Multiprocessors.* Santa Clara Univ., Sept. 1996.

[3] *Proc. Workshop on Communication and Architectural Support for Network-based Parallel Computing (CANPC’97).* http://www.cis.ohio-state.edu/~panda/canpc97.html.

[4] M. Eberl. *Realisierung einer DSM-Implementation von Active Messages am Beispiel eines SCI-gekoppelten Sun Workstation Clusters.* Diplomarbeit, Institut für Informatik, Technische Universität München, Aug. 1996.

[5] T. von Eicken, A. Basu, V. Buch, W. Vogels. “U-Net: A User-Level Network Interface for Parallel and Distributed Computing“. *Proc. 15th ACM Symposium on Operating System Principles.* ACM Press 1995.

[6] T. von Eicken, D.E. Culler, S.C. Goldstein, K.E. Schauser. “Active Messages: a Mechanism for Integrated Communication and Computation”. *Proc. 19th Int’l. Symp. on Computer Architecture.* ACM Press 1992.

[7] T. von Eicken, A. Basu, V. Buch. “Low-Latency Communication Over ATM Networks Using Active Messages“. *IEEE Micro*, Feb. 1995, 46-53.

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R, Manchek, V. Sunderam. *PVM: Parallel Virtual Machine. A User’s Guide and Tutorial for Networked Parallel Computing.* MIT Press 1994.

[9] D.B. Gustavson. “The Scalable Coherent Interface and Related Standards Projects“. *IEEE Micro,* Feb. 1992, 10-22.

[10] D.B. Gustavson, Q. Li. “Local-Area MultiProcessor: the Scalable Coherent Interface“. In: Defining the Global Information Infrastructure: Infrastructure, Systems, and Services, S. F. Lundstrom (ed.), SPIE Press, vol. 56, pp. 131-160, 1994.

[11] H. Hellwagner, I. Zoraja, V. Sunderam. “PVM Data Transfers on SCI Workstation Clusters: Early Experiences”. *Proc. PVM User Group Meeting 1996,* Santa Fe, New Mexico, 1996. http://bay.lanl.gov/pvmug96/proceedings.html.

[12] Special Issue on *Hot Interconnects* (Selected Articles from the *Hot Interconnects II Symposium*, Aug. 1994). *IEEE Micro,* Feb. 1995.

[13] Special Issue on *Developing Interconnect Technology* (Selected Articles from the *Hot Interconnects III Symposium,* Aug. 1995). *IEEE Micro*, Feb. 1996.

[14] M. Ibel, K.E. Schauser, C.J. Scheiman, M. Weis. “Implementing Active Messages and Split-C for SCI Clusters and Some Architectural Implications”. *Proc. 6th Int’l. Workshop on SCI-based High-Performance Low-Cost Computing.* SCIzzL Association, Santa Clara, CA (http://www.SCIzzL.com), Sept. 1996.

[15] M. Leberecht. “A Concept for a Multithreaded Scheduling Environment”. *Proc. 4th PASA Workshop on Parallel Systems and Architectures.*World Scientific Publishing 1996.

[16] S.J. Leffler, M.K. McKusick, M.J. Karels, J.S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley 1989.

[17] *Proc. Second NOW/Cluster Workshop: Building Systems of Systems*. http://www-csag.cs.uiuc.edu/asplos/now-cluster.html.

[18] K. Omang, B. Parady. *Performance of Low-Cost UltraSparc Multiprocessors Connected by SCI.* Research Report 219. Dept. of Informatics, University of Oslo, June 1996. http://www.ifi.uio.no/~sci.

[19] http://www.netlib.org/pvm3/pvm_win32.zip (PVM for the Win32 API).

[20] ANSI/IEEE Std 1596-1992: *IEEE Standard for Scalable Coherent Interface (SCI).* IEEE Computer Society Press 1993.

[21] IEEE Std P1596.9/Draft 0.41: *Physical Layer Application Programming Interface for the Scalable Coherent Interface (SCI PHY-API).* http://sci.lbl.gov.

[22] J. Weidendorfer. *Entwurf und Implementierung einer Socket-Bibliothek für ein SCI-Netzwerk.* Diplomarbeit, Institut für Informatik, Technische Universität München, Feb. 1997.