# Fast Communication Mechanisms – Coupling Hardware Distributed Shared Memory and User-Level Messaging

Hermann Hellwagner, Wolfgang Karl, Markus Leberecht

Lehr- und Forschungseinheit Informatik X,
Rechnertechnik und Rechnerorganisation /
Parallelerechnerarchitektur (LRR-TUM)
Institut für Informatik der Technischen Universität München
Arcisstr. 21, D-80290 München, Germany
Tel.: +49-89-28928278, Fax: +49-89-28928232
Email: {hellwagn | karlw | leberech} @ informatik.tu-muenchen.de

**Abstract** *Low latencies for small messages are an important factor of efficient fine-grained parallel computation. The Active Messages concept provides this minimal overhead by eliminating certain parts of the critical path of sending and receiving messages, that is the context switch into the operating system kernel when using user-mode I/O, and multiple buffering in the network layer. Hardware-supported distributed shared memory (DSM) architectures exhibit various properties that make them particularly useful for an implementation of the aforementioned messaging mechanisms. This paper thus describes the concept, implementation, and the performance of a DSM-based Active Messages layer.*

*Keywords*: Distributed Shared Memory, Active Messages, User-Level Communication, Scalable Coherent Interface

## 1 Introduction and Motivation

Networks of workstations (NOWs) have become increasingly popular as widely available facilities for parallel processing. This has been fostered by the availability of public domain packages like PVM [3] or NXLib [10] as well as implementations of MPI [4], which allow a NOW to be utilized as a virtual parallel computer. In particular, small and medium enterprises, universities, and research labs follow this approach, thereby saving their existing investments in desktop machines. On the other hand, parallel computing with NOWs bears the drawbacks of insufficient communications performance over today's LANs and message passing as the only programming model.

With its technical features and applications, the Scalable Coherent Interface (SCI, ANSI/IEEE Std 1596-1992) [7] offers an approach to solve the problems mentioned above. SCI defines a high-speed and scalable interconnect technology. In addition to I/O style communication, SCI facilitates communication via distributed shared memory. Its protocols provide bus-like services in a fully distributed manner. The shared memory communication allows for low latency data transfers because read or write operations are automatically turned into remote accesses. The increased hardware communication performance can be exploited using a user-level communication layer without the need of invoking system calls, but can also be exploited using load and stores only, into appropriately mapped remote memory regions.

By exploiting the potential of SCI, the SMiLE (Shared Memory in a LAN-like Environment) [5] project at LRR-TUM tries to meet the demands of a low-cost, technically advanced, and powerful parallel computer. As clusters of workstations connected via SCI promise to deliver high performance, we decided to set up such a system with distributed shared memory within the

SMiLE project. We developed and implemented our own PCI-SCI bridge [1] which is targeted to plug into the PCI bus of a PC. Pentium-PCs to be equipped with our PCI-SCI adapter will be interconnected to a cluster of computing nodes with distributed shared memory, the SMiLE multiprocessor system. An Active Messages layer [11] on top of SCI shared memory transactions has been implemented on a testbed of a SCI-cluster of SUN SPARC workstation as an user-level communication layer achieving extremely low-latencies. Later on, it will be ported on the SMiLE multiprocessor system.

The paper gives an overview of the SMiLE project at LRR-TUM and its objectives. It describes the architecture of the PCI-SCI bridge and presents first performance results. We describe our Active Messages implementation and show that this communication architecture only adds an insignificiant amount of latency to the raw latency of SCI.

## 2 The SMiLE Multiprocessor System

The SMiLE multiprocessor to be built up with industry-standard PCs interconnected by SCI technology will serve as main research vehicle. The long-term objective of research is to design and implement an appropriate distributed shared memory programming model and to investigate the efficient use of a parallel computer with NUMA characteristics (non-uniform memory access).

We started our own hardware and system software developments because commercial available PCI-SCI adapters do not provide the flexibility and extendability required for our research (e.g. monitoring functionality).

We have developed a PCI-SCI bridge, which serves as the interface between the PC's I/O bus, the PCI bus, and the SCI network. SCI nodes are interconnected via the input and output links, so that ring-like connections are built with the output of one node providing the input to the input link of the neighbour node.

The SCI standard defines packet switched communication protocols. SCI split transactions require a request packet to be sent from one SCI node to another node and a response packet transmitted from the remote node back to the source node in order to complete the transaction. The PCI-SCI interface generates packets for remote SCI nodes, transmits incoming packets via the output link to the neighbour SCI node or directs them to the local user node. PCI address spaces of the bridge can be mapped into SCI, allowing PCI read/write accesses to any of the mapped SCI resources. The PCI-SCI interface is responsible for potentially required address translations and request/response packet generation. Such a transparent access is the key feature required for implementing shared memory applications.

As depicted in Figure 1, the PCI-SCI bridge is divided into three logical parts, the PCI unit, the Dual-Ported RAM (DPR), and the SCI unit. The PCI unit interfaces to the PCI bus of the local PC host, and the SCI unit interfaces to the SCI network. Data is transferred between the PCI unit and the SCI unit across the Dual Ported RAM (DPR), and control information is passed on the handshake bus. For the interface to the PCI bus, the PCI9060 chip from PLX Technology is used, which translates read/write operations into internal bus operations and vice versa. The PCI unit is connected to the DPR via a 32 bit wide internal bus, the DPR bus. The DPR and the SCI unit are connected via the B-Link, a 64 bit wide synchronous bus. The B-Link is the back side of the interface chip to the SCI network, the Link-Controller LC1 from Dolphin. The Link-Controller LC1 implements the physical layer and parts of the logical layer of SCI. Two functional units are responsable for the coordination and communication between the PCI and SCI units. These two functional units are implemented for chips of the Xilinx 4000E FPGA series.

### 2.1 The PCI Unit

The PCI unit (Figure 2) has to translate PCI read/write transactions into bridge internal bus operations and vice versa. For this task, the PCI9060 chip from PLX Technologies [9] is used, which provides a PCI bus master interface for adapters.

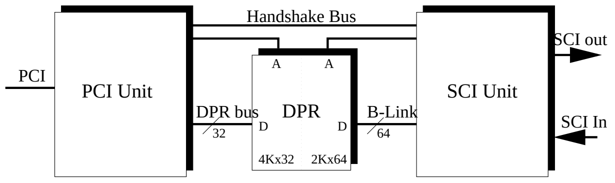

Figure 1: PCI-SCI bridge architecture

The chip's local bus follows the Intel i960 microprocessor's bus protocol. Using the i960 bus as the PCI unit's local bus simplifies the design of the control logic considerably. No complex PCI compliant control logic is necessary. The PCI9060 supports both multiplexed and non-multiplexed local buses. Operating with the multiplexed bus mode for addresses and data can be exploited for the generation of SCI packets. The PCI9060 can act both as target and as initiator on both of its sides, thus supporting bidirectional transactions forwarding. Two independent bi-directional DMA channels are integrated, allowing direct memory transfers between PCI and SCI initiated by the PCI-SCI bridge. The PCI9060 local bus interface runs from a local TTL clock and generates the necessary internal clocks. This clock runs asynchronously to the PCI clock. This feature simplifies the implementation, because the internal bridge bus can operate at a different speed than the PCI bus.

The SCI Upload and Packet EncodeR Management Unit (SUPER_MAN) is responsible for controlling and coordinating the translation of read/write accesses into B-Link packets and vice versa. Simultaneously, the ATC devices provide address translation from PCI's 32-Bit addressing scheme to SCI's 64-Bit addressing while the dual-ported RAM (DPR) serves as the main buffer and scratchpad memory for outgoing and incoming transactions.

## 2.2 The SCI Unit

The SCI Unit (Figure 3) is connected to the DPR via the B-Link. The B-link Access and Transaction MANager (BAT_MAN) controls B-Link arbi-

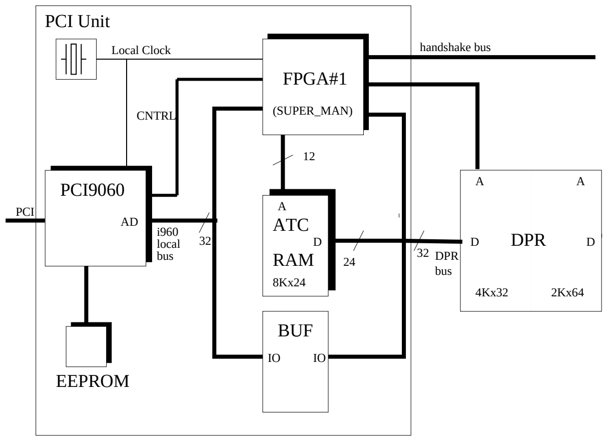

Figure 2: Block diagram of the PCI Unit

tration, reading or writing the DPR (on SCI unit side), and drives the B-Link control signals. It is also implemented on a FPGA chip.

The physical layer and part of the logical layer of SCI are implemented by Dolphin's Link Controller (LC). The LC provides the SCI input and output links on the SCI side. On the "back side" (non-SCI link side) of the chip is the B-Link.

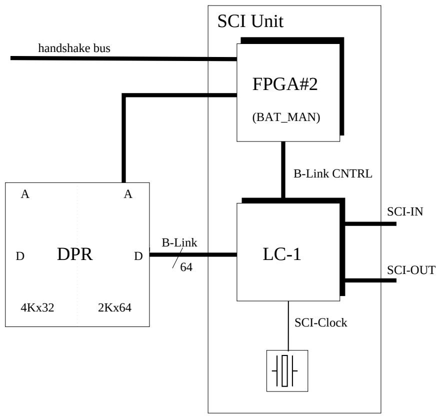

Figure 3: Block diagram of the SCI Unit

The development of the PCI-SCI bridge has been completed and the prototype is now being tested. For the measurements, the prototype operates in a loopback mode. The system clock currently runs at 18 MHz, while the SCI network runs at 50 MHz. The latency for a write transaction is 3.1 $\mu$s. The bridge is being tuned to run at a system clock of 25 MHz. Thus, we expect performance to improve by about 40 %. With the DMA block transfer feature of the PCI-SCI bridge, a bandwith of about 10 MB/s can be achieved when transfering 1024 Bytes (in 16 64bytes packets) at a clock rate of 12 MHz. When running at 25 MHz, a bandwidth of about 20 MB/s can be expected for a single node. The bandwidth of an SCI ring is 125 MB/s (1 Gb/s). In the final version of the paper we will be able present the performance data of an optimized design.

The SMiLE multiprocessor will be built using the PCI-SCI bridges. For a running system, additional work has to be performed. This work comprises the development of a device driver for Linux providing transparent memory accesses and message passing style communication, and the development of a device driver for Windows NT.

In the next section, we present an Active Messages layer implementation for low-latency, user-level communication layer for the SMiLE multiprocessor system.

## 3 Active Messages and SMiLE

### 3.1 Active Messages and User-Level Communication

Analysing the high message setup times in common communication packages on parallel and distributed systems, researchers at UC Berkeley discovered that a relatively high amount of copying is responsible for not making use of increased hardware communication performance [11]. The associated strategies of multiple buffering allow for complex protocol stacks, asynchronous messaging, and mutual protection of concurrent processes using the communication layer.

*Active Messages* reduce the setup latency of sending and receiving messages by one or several orders of magnitude on some systems. This is achieved through a different understanding of a message which not only contains the relevant data but also a function pointer indicating a so-called handler function. Upon receipt of a message, this function is called with the transported data as its parameter, thus effectively removing the message from the communication layer. Through this technique, buffering is reduced to the absolute minimum and thus allows for minimal latencies.

Architectures that allow user-level access to the network interface (NI) benefit the most from the Active Message concept: The second source of increased latency in common communication layers is the context switch into the operating system kernel, necessary to maintain synchronization and protection of resource accesses among the processes. If these tasks are instead performed by a user-level communication layer it is possible to allow for all properties of a conventional communication layer while achieving extremely low latencies.

The Scalable Coherent Interface is such a communication architecture allowing for user-level access not only to the network interface but–through remote memory transactions–direct access to remote node's local memories. The Active Message layer within the SMiLE project attempts to close this possible security loophole by implementing the Active Message specification 2.0 [8] which extends the original definition with the notion of communication endpoints and bundles through which protection and shared access to the network interface are provided.

By this definition, an *endpoint* consists of

- a send pool for outbound messages,
- a receive pool for incoming messages,
- a handler table mapping function pointers to handler indizes,
- a virtual memory segment of the application for bulk transfers, and
- some additional management information.

An *endpoint bundle* is the combination of one process' several endpoints and consists of

- a synchronization variable, indicating an endpoint's communication event,

- a suitable event mask, and
- an access mode flag (concurrent or sequential) indicating the way the endpoints or the bundle are used.

### 3.2 The SMiLE Active Message Layer and its Structure

The SMiLE Active Message Layer essentially has two components:

- The *Active Messages daemon*, launched on every physical node of the system and responsible for creating, initializing, and mapping of the shared memory areas that are provided to AM libraries, and other daemons.
- The *Active Messages (AM) library*, linked to user applications, it contains all data types and functions of the AM application programmer's interface. Send and receive pools are naturally being implemented as shared memory segments.

Shared memory transactions allow for a possible optimization of an Active Message system: While remote write accesses can be immediately acknowledged to the sender and buffered in the network interface until completion, the same does not hold for remote read accesses during which the processor has to wait for completion of the transaction and delivery of data. It is thus favourable to implement buffers that are shared memory pages on the receiving side of a connection, effectively using only buffered remote writes as a means of transporting data. SMiLE's Active Message layer is therefore structured as depicted in Fig. 4.

Each receive pool is constructed as a ring buffer of a configurable number of messages in order to allow for non-blocking send operations of a multitude of small messages.

Since common SCI hardware does not yet allow remote signalling, polling was the only way of notifying the receiving side of an incoming message. Its drawback of a short period of blocking the receiver node's processor is offset by the economic use processing time: an application only works

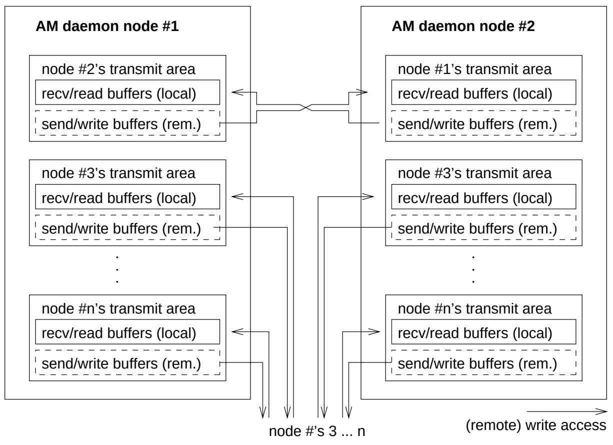
AM daemon node #1
AM daemon node #2
node #2's transmit area
recv/read buffers (local)
send/write buffers (rem.)
node #1's transmit area
recv/read buffers (local)
send/write buffers (rem.)
node #3's transmit area
recv/read buffers (local)
send/write buffers (rem.)
node #3's transmit area
recv/read buffers (local)
send/write buffers (rem.)
node #n's transmit area
recv/read buffers (local)
send/write buffers (rem.)
node #n's transmit area
recv/read buffers (local)
send/write buffers (rem.)
node #'s 3 ... n
(remote) write access

Figure 4: Structure of SMiLE's Active Message layer

on "receiving" messages (checking receive pools and executing handler functions) as soon as it finds time for it.

## 4 Experiments and Analysis

Several experiments with this setup have been performed and show the expected behaviour. The Active Message layer only adds an insignificant amount of latency to the raw latency of SCI, while the unoptimized overall bandwidth amounts to more than 80 % of the network's raw bandwidth. (See Table 1.) While the figures of the AM version 2.0 are due to our own experiments on a cluster of eight Ultrasparc Workstations interconnected through Dolphin SBus-2 SCI adapters (four ringlets of two nodes attached to bus-based switch), the AM version 1.1 results on SCI, the Meiko CS-2, and a Myrinet NOW were taken from [6] while the ATM results are taken from [2].

A drawback of the current SCI-focused implementation should be mentioned, however, which also holds for other hardware-DSM versions of this communication paradigm. The fact that the sending node has to determine the location at the receiving site to which it will store the data, makes it harder to scale the system: In order to keep latencies low and avoid costly synchronization of a

Table 1: Comparison of roundtrip latencies and transfer rates (AM Store operation) for different Active Messages implementations

| Machine | RT Lat. | BW |
|---|---|---|
| SCI NOW / AM 2.0 (switched) | 15 $\mu$s | 21 MB/s |
| SCI NOW / AM 1.1 (switched) | 16.5 $\mu$s | 13 MB/s |
| Meiko CS-2 | 23 $\mu$s | 32 MB/s |
| Myrinet NOW | 28.9 $\mu$s | 34 MB/s |
| U-Net (ATM NOW, True Zero Copy) | 71 $\mu$s | 14 MB/s |

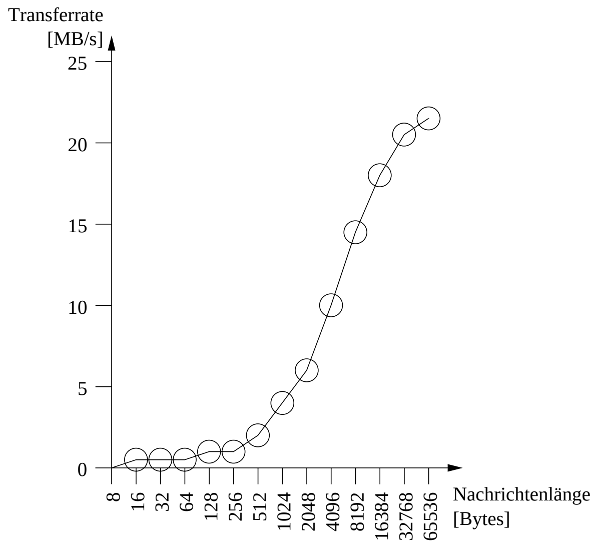

Figure 5: Message throughput versus message size for AM Store operations

shared receive buffer, it is necessary to partition buffer space among multiple connections. This effectively forces the receiving node to scan every possible buffer at the reception of a message. Consequently, as can be seen from Fig. 6, the amount of time spent for checking the buffers grows linearly with the number of nodes.

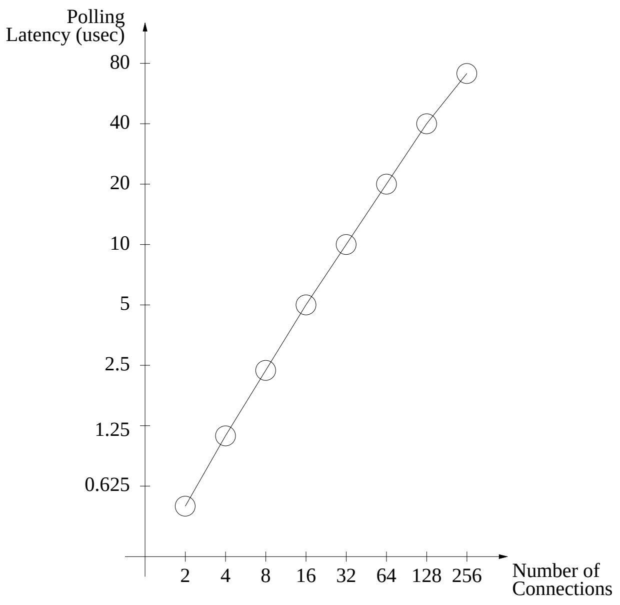

Figure 6: Polling-time for several end points

Possible solutions to this problem require the receiving side to determine the buffer location of a new message. In [6] the inclusion of another remote memory transaction to current hardware-DSM systems is therefore suggested, namely the addition of a *Remote Enqueue* operation. This operation inserts a data packet at the end of a queue in remote memory, by this alleviating the sender from the duty to determine an exclusive buffer space at the receiver's end while allowing the latter to merely check a single memory location instead of scanning a number of buffers that is proportional to the number of connections.

## 5 Conclusions and Outlook

Our experiments have clearly shown that Active Messages and user-level communication in a hardware-supported DSM system perform well and are worth being considered as part of a general message-oriented communication layer. Scalability problems inherent in shared memory based communication can be overcome by yet to be included special remote transactions.

With our own PCI-SCI bridge, we are able to build the SMiLE multiprocessor, a low-priced parallel computer consisting of standard PCs forming an efficient NUMA DSM system. With its software layers, particularly the Active Messages layer, it is possible to perform message-passing-style communincation that benefits largely from its hardware supported distributed shared memory and the possibility of direct user-level access to the network interface.

Thus, it will subsequently provide us with a platform for experimental research and investigations into the efficient use of DSM parallel machines and their accompanying programming models.

## Acknowledgements

We would like to thank Klaus Schauser and his research group at the University of California at Santa Barbara for access to their network of SCI-coupled workstations which only made the Active Messages experiments possible.

## References

[1] Georg Acher, Hermann Hellwagner, Wolfgang Karl, and Markus Leberecht. A PCI-SCI Bridge for Building a PC-Cluster with Distributed Shared Memory. In *Proceedings The Sixth International Workshop on SCI-based High-Performance Low-Cost Computing*, pages 1–8, Santa Clara, CA, September 1996. SCIzzL.

[2] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, Colorado, December 1995. ACM.

[3] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, Massachusetts, 1994.

[4] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI – Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts, 1994.

[5] Hermann Hellwagner, Wolfgang Karl, Markus Leberecht, Harald Richter, and Vaidy S. Sunderam. SCI-Based Local-Area Shared-Memory Multiprocessor. In *Proceeding APPT'95 – Int. Workshop on Advanced Parallel Processing Technologies*, pages 32–39, Beijing, China, September 1995. Publishing House of Electronics Industry.

[6] Maximilian Ibel, Klaus E. Schauser, Chris J. Scheiman, and Manfred Weis. Implementing Active Messages and Split-C for SCI Clusters and Some Architectural Implications. In *Proceedings The Sixth International Workshop on SCI-based High-Performance Low-Cost Computing*, pages 39–48, Santa Clara, CA, September 1996. SCIzzL.

[7] IEEE Standard for the Scalable Coherent Interface (SCI). IEEE Std 1596-1992, 1993. IEEE 345 East 47th Street, New York, NY 10017-2394, USA.

[8] Alain Mainwaring and David Culler. *Active Messages: Organization and Application Programming Interface*. Computer Science Division, University of California at Berkeley, November 1995.

[9] PLX Technology Inc., 625 Clyde Avenue, Mountain View, CA. *PCI 9060 PCI Bus Mas-*

*ter Interface Chip for Adapters and Embedded Systems*, April 1995. Data Sheet.

[10] Georg Stellner, Arndt Bode, Stefan Lamberts, and Thomas Ludwig. NXLib - A Parallel Programming Environment for Workstation Clusters. In C. Halatsis, D. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE'94 Parallel Architectures and Languages Europe*, number 817 in Lecture Notes in Computer Science, pages 745–748. Springer-Verlag, 1994.

[11] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, volume 20 of *CAN*, pages 256–266, Gold Cost, Australia, May 1992. ACM.