The Scalable Coherent Interface (SCI) as an Alternative for Cluster Interconnection

César A. F. De Rose[1], Reynaldo Novaes[1], Tiago Ferreto[1], Fábio A. D. de Oliveira[2], Marcos E. Barreto[2], Rafael B. Ávila[2], Philippe O. A. Navaux[2], Hans-Ulrich Heiss[3]

[1] High-Performance Computing Center
Catholic University of Rio Grande do Sul (PUCRS), Brazil
E-mail: {derose,novaes,ferreto}@cpad.pucrs.br

[2] Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS), Brazil
E-mail: {fabreu,barreto,avila,navaux}@inf.ufrgs.br

[3] Fachbereich Informatik
TU-Berlin, Germany
E-mail: heiss@tu-berlin.de

Abstract—

The SCI standard was proposed in 1992 to be a high performance bus for processor interconnection in multicomputers. With the growing popularity of cluster architectures, the standard was implemented in PCI cards and offered as a flexible an efficient alternative for the construction of parallel systems. After some initial difficulties, like incompatibilities with various PCI chipsets and the absence of drivers for some platforms, the SCI-based products are already mature but the expected breakthrough is not yet reached. In this paper we present our experience in the construction and utilisation of SCI clusters and in the development of support tools and applications for these systems. We present a survey of the obtained results over the last years in this area, including our own research, and make an analysis on how the standard and its implementations are performing today.

Keywords— **Distributed shared memory, cluster computing, SCI.**

I. Introduction

Since the adoption of cluster-based architectures as execution platforms for parallel and distributed applications, starting in the middle of the 90's, the development of communication protocols and programming environments has been a common activity for many academic and industrial research groups, whose goal is to efficiently exploit the resources provided by the underlying communication networks used in this kind of architecture. The most popular example of such communication networks today is Myrinet [BOD 95].

From this effort, a great number of low-level communication protocols such as BIP [PRY 98] and GM [GM 99] have arised with the goal of providing a set of primitives to allow the user to specify the communication behaviour of his application, taking advantage of the low latency and high bandwidth rates provided by the communication networks. Besides, several implementations of the MPI standard [MPI 94] on top of these communication protocols have been done.

SCI—Scalable Coherent Interface [IEE 92]—differs from the other communication technologies applied to cluster computing due to its shared memory facilities. This feature stimulates a high-level communication abstraction, based on shared segments and remote load/store operations; at the same time, it imposes some constraints on the communication protocol as well as the applications regarding the management of these shared segments and also message passing.

In this paper we present an overview of the SCI communication standard, regarding its features, constraints and low-level APIs. Besides, we discuss its utilisation on the development of parallel programming environments and some communication protocols with message passing and DSM semantics currently available.

The results presented in this paper[1] have been obtained from over 5 years of study and utilisation of SCI as a communication technology for the clusters available to our research groups [ÁVI 99, ÁVI 00]: the GPPD (Parallel and Distributed Processing Group) from UFRGS, the CPAD-HP (High Performance Computing Centre) from PUCRS and the PC^2 (Paderborn Centre for Parallel Computing) from the University of Paderborn, in Germany. The three groups are pioneers in the use of SCI within the Brazilian and German communities, having developed low-level software, programming environments and published several papers regarding this subject. The three Universities maintain, since 1999, an International Cooperation Project within the CAPES/DAAD programme on the development of tools for SCI.

In this manner, our goal with this paper is to bring to the scientific community our knowledge using this architecture to the execution of parallel and distributed applications and to the development of programming environments, in which questions related to the management of shared segments and message passing must be addressed if the goal is to provide the user with high performance and transparency.

This paper is organised as follows: Section II presents an

[1] Results for DECK have been measured at the GPPD/UFRGS; results for Yampi have been measured at the CPAD-HP/PUCRS

overview on the SCI technology and its main features; Section III brings a survey on many different tools available for SCI programming; in Section IV we present the software developed by our groups and some of the obtained results up to now; and finally Section V brings the authors' final considerations.

II. The SCI interconnect

The *Scalable Coherent Interface* (SCI) is an IEEE standard that provides computer-bus-like services to a set of nodes via fast unidirectional links connected in a ring. SCI uses a point-to-point interface between the network nodes, which allows several topologies like rings, meshes, multistage networks and crossbars to be chosen. The ring topology, however, is especially suitable since it is very simple and inexpensive to realize. The SCI standard specifies the supported interface to run at 500MHz over 16 parallel signals yielding a raw point-to-point throughput of 1GB/s.

An interesting characteristic of available SCI cards is the native 2- and 3-d support; Figure 1 shows the topology of PSC-64, a 64-processor SCI cluster available at Paderborn. Notice that no switch devices are needed since the 2-d cards allow the construction of mesh-like topologies. In the case of PSC-64, there are 32 bi-processor machines connected in a 4×8 mesh, as can be seen on the Figure. Paderborn still owns a 192-processor SCI cluster (PSC-192), forming a 8×12 mesh.

SCI inserts a third type in between the *loosely coupled* and the *tightly coupled* multiprocessor systems: the *snugly coupled system* [DOL 00], characterising the cluster as a non-uniform memory access machine (NUMA). This type of interconnection network combines the benefits of shared memory systems with the reliability advantages of the loosely coupled systems. In a snugly coupled system, each node has a private copy of the operating system, whereas the application is distributed across the nodes. Node failures do not halt the entire system and management software can cope with the interruption. At the same time, the application continues to have access on shared memory spread across the nodes.

A SCI cluster therefore does not only provide the facilities for message passing communication, but also enables parallel programs to use shared memory segments. Unfortunately, the PCI-based implementation has, unlike the original IEEE standard, a major drawback: the idea of the standard is to have a cache coherent system which spreads over the whole cluster onto many nodes; the caching of remote memory is, however, not possible for PCI-based systems, since transactions on the main bus of a local node are not visible on the PCI bus. Thus, a PCI-based card like the Dolphin SCI card used in this paper cannot take part in the coherence protocol on the main bus.

III. Available work

In this Section we present a panorama of the available work on programming APIs for SCI. We have divided this presentation in three main groups: APIs following a shared-memory model, APIs following message-passing and low-level APIs primarily intended for support.

A. Shared memory

Since the whole idea of SCI is to provide shared memory, it is natural to think of programming environments which exploit this approach. The main differences presented by such environments lie on the transparency level in relation to shared memory, as shall be seen next.

A.1 SMI

SMI (Shared Memory Interface) [DOR 99] is a programming library for SCI developed at the RWTH, Aachen, Germany, which provides primitives for the establishment and sharing of distributed memory segments. The application runs over a given number of nodes, based on a SPMD model.

Processes in SMI communicate with each other by mapping remote memory segments created on other nodes. Shared segments compose *shared regions*. Three policies for the placement of a region are available: *undivided*, which means that a single segment composes the region, being entirely located in a single node; *blocked*, in which a region is composed of multiple segments, which in turn are uniformly distributed among the nodes of the application; and *customised*, in which the user if free to decide how many segments compose a region and where they should be placed.

An interesting characteristic of SMI is that the user, for performance purposes, may decide to replicate a given segment throughout the nodes, thus avoiding remote accesses. Of course, it is his responsibility to guarantee that distinct nodes do not modify the same area, otherwise consistency problems may arise. Later, he may decide to turn off the replication mechanism, in which case SMI merges the modifications into a single copy.

For synchronisation, SMI offers barriers, mutexes and progress counters. The first two follow the conventional semantics; progress counters may be used by a process to indicate its computational progress during execution, so that other processes can be aware of this information.

A.2 SCI-VM

SMI follows the approach naturally enforced by SCI, namely that of shared-memory programming. But still the programmer is forced into executing tasks not common to ordinary shared-memory programming, such as the explicit creation of a segment to be shared. The keyword in this issue is *transparency*; and this is the goal of SCI-VM, or SCI

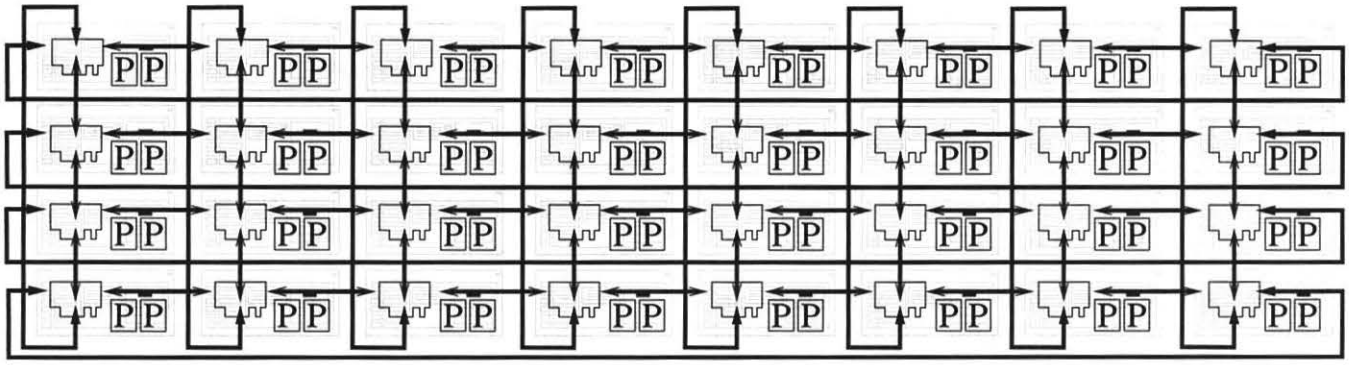

Fig. 1: Example of a 2-d SCI cluster.

Virtual Machine [SCH 99]. This project is being carried out in Munich, Germany, and consists on the establishment of a global logical address space over all the nodes of a SCI cluster; in this way, it should be possible to achieve completely transparent shared memory, and a SCI cluster could be treated as a single SMP computer. In fact, the group is also working on a Pthread-like environment, named SISCI-Pthreads, to run globally over the SCI nodes.

B. Message passing

Being a high-performance communication technology, it is natural that standard message-passing environments have also been implemented for SCI-based parallel machines. This Section presents a representative set of such environments.

B.1 ScaMPI

ScaMPI [HUS 99] is a MPI implementation on top of SCI developed by Scali AS, a Norwegian company pioneer in commercial high-performance systems based on SCI. A number of goals have been set in the implementation of ScaMPI: scalability, low latency, high bandwidth, fault tolerance, flexibility of transport medium (e.g. to easily use system memory instead of SCI for local communication), user friendliness and thread-safeness. All of these are said to be fully achieved by Scali, most of which could also be verified by our groups.

Some specific features of ScaMPI are needed in order to carry out the desired advantages; for example, a checkpointing mechanism is used to guarantee the atomicity of all SCI data transfers. User-friendliness is a result of the tool `mpimon`, a command-line application used to start and control the execution of a ScaMPI application. While the traditional `mpirun` is more intuitive, the use of `mpimon` allows for additional execution parameters as well as the integration with Scali's management software.

In terms of performance, ScaMPI is able to deliver 9.4μs latency and 76MB/s bandwidth between two Pentium II 450MHz nodes, which is considered quite satisfactory. Besides, ScaMPI is thread-safe, so the application can benefit from SMP machines by making use of additional threads of control.

B.2 SCI-MPICH

The same research group from Aachen responsible for the implementation of SMI has also developed SCI-MPICH [WOR 00]. Normally, MPICH only demands the adaptation of its ADI (Abstract Device Interface) in order to accomplish a new port; thus, in the case of SCI-MPICH, an ADI for SMI has been implemented.

To achieve good performance, SCI-MPICH makes use of three different message exchange protocols, based on the payload size: *short*, *eager* and *rendez-vous*. This enables the obtention of around 7μs latency, when the protocol *short* is in action, and over 73MB/s bandwidth with the *rendez-vous* protocol.

B.3 PVM-SCI

One of the two available implementations of PVM for SCI is PVM-SCI [FIS 99]. A differential characteristic of this implementation is that it provides two complementary communication mechanisms: conventional TCP/UDP protocols for sending messages from one task to another via the PVM daemons, and a *ad-hoc* protocol which makes use of SCI when the `PvmRouteDirect` option is set.

The architecture of PVM-SCI has been kept modular to allow for easy adaptation to other technologies (e.g. Myrinet). It initially tests for the availability of interconnect adapters; if SCI is not present, the application is run on the traditional (usually present) Ethernet connection.

This implementation is one of the few that use the SCI interrupt mechanism for delivery of messages. As a result, since this mechanism is too slow when compared to SCI's low latency, the performance of PVM-SCI is still poor.

Raw measures indicate a minimal latency of about 42μs, and a peak bandwidth of 14MB/s. The use of ordinary `memcpy()` instead of MMX or floating-point instructions also contributes for these figures.

B.4 SCIPVM

Another PVM implementation for SCI is SCIPVM [ZOR 99]. To the difference of the previously presented implementation, SCIPVM offers a non-intrusive approach to using the SCI network: two new functions `pvm_scisend()` and `pvm_scirecv()` have been added, which expect the same arguments as those of PVM's original functions. These functions take effect when direct routing is used.

SCIPVM has been originally designed for SCI interfaces available for Sun workstations, which do not include the stream buffers present on PCI interfaces, so a direct comparison with the other environments would not be appropriate. Like PVM-SCI, this implementation also makes use of hardware interrupts to signal the completion of I/O operations, which is too slow to result in good performance.

C. Active messages

Active Messages [EIC 92] is a low-latency communication mechanism. Each active message contains the address of a handler function which is executed on the receiving processor upon arrival of the message. Message handlers are intended to be short and execute quickly. Under the AM model, messages travel from user space (the send instruction) directly to user space (the message handler), avoiding any form of buffer management and synchronisation usually encountered in the traditional send/recv model. As a result, AM can achieve an order of magnitude performance improvement over more traditional communication mechanisms.

The SCI AM [IBE 96] is based on the *Remote Queue Abstraction*. Sending processors just enqueue their message on the remote queue. During a poll, the receiving processor just checks whether something has been enqueued. If so, it removes the message from the queue and processes it. This remote queue abstraction can be built easily on traditional message-passing network interfaces, because they have a single entry point which essentially acts as a queue. It is the receiving processor's responsibility to poll the messages from the network interface and process them.

D. Low-level APIs

D.1 SISCI

The SISCI API [GIA 98] is an effort from both the academia and the industry to establish a set of standard primitives for basic SCI programming. Dolphin, for example, engaged the European ESPRIT Project with the goal of achieving an API for clustering and the SISCI API follows this standard.

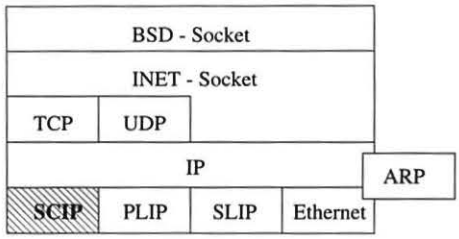

Fig. 2. Linux networking layers.

It consists of a complete shared-memory API that allows an application running locally to operate on remote memory segments in user space. Together with driver software, adapter cards and switches, SISCI enables applications to bypass the traditional networking protocol limitations, minimising time-consuming operating system calls, and heavy networking software overhead due to hardware support.

Because it is strongly hardware-oriented, and therefore very powerful, SISCI is often used as building blocks for other, more user friendly API's like ScaMPI and SMI.

D.2 CML

CML, or Common Messaging Layer [HER 98], is a set of low-level primitives especially designed to support implementations of MPI and PVM. CML is being developed within the context of the SISCI project just mentioned.

This library does not introduce any new techniques in terms of message passing for SCI; it tries to avoid remote readings and local memory copies, which are two of the most influent issues in relation to communication performance. An initial implementation has been able to achieve 14μs latency and around 35MB/s bandwidth.

D.3 SCIP — Scalable Coherent Interface IP

Instead of replacing TCP by a user-level transport protocol to use the SCI interface, SCIP [TAŞ 98] is a network driver which works underneath the IP layer (Figure 2). It is transparent for socket programmers, because it keeps the socket semantics, i.e. all applications work without any modifications or recompilations.

SCIP is implemented as a Linux module. The most important advantages are the maintenance of socket semantics, and the fact that it can be used by kernel services (NFS, routing) and it runs without problems on multiprocessor machines since the Linux kernel provides coordinated access to the driver.

The main problem of SCIP, as usual for kernel-level communication software for clusters, is its high overhead. The

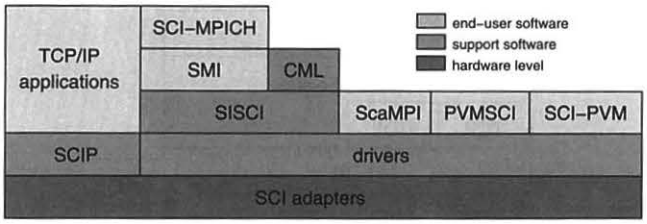

Fig. 3. Location of each of the presented projects within the software stack.

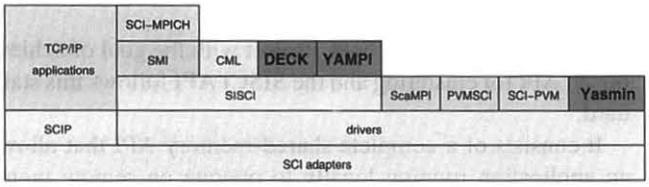

Fig. 4. Yasmin, Yampi and DECK in the software stack.

latency measurements of SCIP reveal 77μs latency. Nevertheless, SCIP shows that using SCI at the lowest layer in the protocol stack works well and has the advantage that all existing applications work as well. It is conceivable to use SCIP as a substitute for the existing Ethernet network in SCI clusters. Another important fact is that, although SCI does not guarantee correct delivery of programmed I/O operations, it ensures that in this solution TCP/IP will correct these errors. This makes it possible to use the simple ring buffer without flow control.

E. Summary

In general, the projects just presented have a common objective which is to bring the SCI capabilities as best as possible to the programmer level. While shared-memory programming libraries do exist, one notices a stronger impact for message passing environments, especially MPI, which is still the *de facto* standard in the field of parallel programming.

Figure 3 shows a software stack where all the mentioned libraries and environments can be located in relation to user, kernel and hardware level.

IV. Contributions from our groups

In this Section we present the software developed up to now by our research groups aiming at SCI programming. The three projects, Yasmin, DECK and Yampi also fit in the previously established division, namely shared memory, message passing and low-level APIs, in that order. Figure 4 shows how our contribution can be added to the already shown software stack.

A. Yasmin

Yasmin (Yet Another Shared Memory Interface) [TAŞ 98a] is a programming environment similar to SMI, developed at the University of Paderborn, Germany. It follows the same idea as SMI, in the sense that the basis for communication is the explicit sharing of memory segments. To the difference of SMI, Yasmin does not provide the concept of shared regions, but presents different features such as collective communication.

Yasmin proposes a mechanism of *groups of processes*, so that memory segments are shared among the processes which make part of a group, and not necessarily among all the nodes of an application. This may give the programmer more flexibility when different communication paths need to be established.

As in SMI, synchronisation is achieved by means of barrier and mutex mechanisms, but Yasmin also provides *signalling objects*, which act as condition variables. A signalling object is associated to the primitives *wait* and *signal*.

Collective communication is implemented with primitives similar to those of MPI (i.e. bcast, scatter, gather, and others), following exactly the same semantics in order to facilitate its understanding.

B. DECK/SISCI

DECK (Distributed Execution and Communication Kernel) [BAR 00, BAR 00a] is a parallel programming environment developed at the GPPD/UFRGS, intended for the programming of clusters. Early versions of DECK have been implemented, first based on Unix sockets, and then on BIP for Myrinet [PRY 98, BAR 00b]. With the acquisition of a SCI cluster by the group, a SCI implementation has been provided [OLI 01].

Communication in DECK is realised by means of two kinds of abstractions: *messages* and *mail boxes*. Messages are containers in which data can be packed and unpacked; mail boxes, in turn, are temporary place-holders for messages sent from one node to another. In order for two distinct threads to communicate, a mail box must be created by one of them, and given a well known name. The peer thread must then execute a *clone* primitive, passing the name as an input parameter, and receiving a reference to the remote mail box when the function returns.

The actual message exchange is performed by the two primitives *post* and *retrv*, semantically equivalent to traditional *send* and *recv* operations. In terms of synchronisation, a *post* is always asynchronous, while a *retrv* is always synchronous.

Some of DECK primitives related to communication are shown below:

```
deck_init()
```

```
deck_done()
deck_mbox_create(&mbox, name)
deck_mbox_clone(&mbox, name)
deck_mbox_post(mbox, msg)
deck_mbox_retrv(mbox, &msg)
```

B.1 The SCI implementation

Two main goals have been defined for the implementation of DECK for SCI: it should keep the same API already presented on the previous implementations, and it should be able to exploit the maximum achievable performance of the underlying network.

The access to the SCI network is performed by using the SISCI API; this approach represents a balanced choice between flexibility (and thus performance) and ease of use.

Similarly to SCI-MPICH, DECK/SISCI also makes use of three different message-exchange protocols in order to maximise communication performance:

- protocol "1" is responsible for small messages, ranging from 0 to 62 bytes; in this case, independently from the actual size of the message, a single 64-byte SCI packet is sent across the network, which optimises the use of the stream buffers in the SCI adapter and thus results in optimal performance
- protocol "2" covers the range of messages from 63 bytes to 8KB. Each mail box is associated to a set of buffers where messages are stored upon arrival; naturally, such buffers consist of shared segments which are directly accessible from remote nodes
- protocol "3" is activated for large messages; in this case, a handshaking is used in order to obtain a zero-copy message transfer. This means that a *post* operation will wait for the complementary *retrv* to occur, in which case the user buffer will be known to DECK and the message can be transferred directly to that location.

B.2 Performance

The graph on Figure 5 shows the raw performance (latency and bandwidth) obtained with DECK/SISCI on a simple ping-pong application, run on the SCI cluster available at UFRGS (4 × Dual Pentium III 500MHz). It can be observed that DECK is able to reach practically the full performance of SCI for PCI adapters, presenting a maximum bandwidth of over 83MB/s; the corresponding latency time for 0-byte messages lies in the range of 4–5μs.

C. Yampi

Yampi (Yet Another Message Passing Interface) is a MPI subset that was developed to give the user a simple message passing interface with a performance level close to the capabilities of the underlying SCI hardware.

It is natural that the overall design goal was efficiency and,

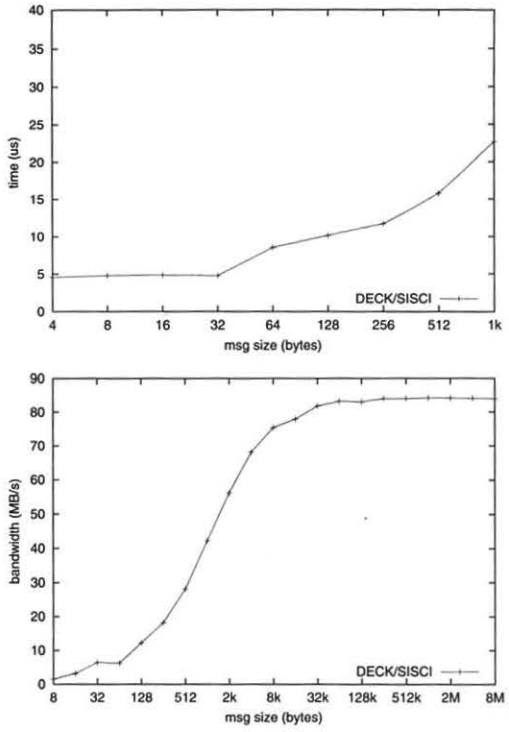

Fig. 5. Raw performance of DECK/SISCI.

to achieve these, two subgoals were defined:

- Efficiency: the design of Yampi is entirely done with SCI capabilities to achieve performance values near the raw memory transfer over the SCI network, between 3 and 5 μs. The ScaMPI implementation has latency values around 12 μs in the tested platform and our goal is to reach values between 5 and 7 μs
- Thread safeness: in order to ensure the correct behaviour of the Yampi in multi-thread environments, locks will be introduced. Since thread safeness is not an issue for some applications, this feature will be optional to avoid the extra time needed for locking

C.1 Functionality

The first version of Yampi will only implement the basic MPI functionality and is expected to be available on October 2001. Yampi will have around 10 routines. The main ones are listed below:

```
YAMPI_Init
YAMPI_Finalize
YAMPI_Comm_Rank
YAMPI_Comm_Size
YAMPI_Send (blocking send)
YAMPI_Recv (blocking receive)
YAMPI_Isend (non-blocking standard send)
```

`YAMPI_Irecv (non-blocking standard receive)`

C.2 Structure

To make Yampi more portable it has been built on top of the SISCI API. It currently supports Linux on Intel platforms and offers C bindings. SISCI is required to establish globally shared SCI memory segments mapped onto a process' address space.

C.3 Initial results

In this section some preliminary performance characteristics of a point-to-point routine are presented. The tests are performed on a single platform: a 4-node cluster using high-end PCs equipped with single Pentium III processors at 550MHz using 440BX/ZX chipset under Linux, directly connected through Dolphin PCI/SCI adapters cards. One single test has been performed: a ping-pong test to measure the round-trip time (two-way latency) for messages of different sizes. The Yampi tasks were started on two different nodes. Each task executes a blocking send and receive operations to wait for an incoming message (Yampi_Recv()) and immediately responds (Yampi_Send()) once the message arrives. Of course, this simple tests to measure the latency and bandwidth of blocking send and receive operations between two processes is not a complete metric for performance of an MPI implementation. It is rather used to give a performance evaluation of the core functionality.

The results are shown in Figure 6. One can notice that the measured latency is higher then the expected range (5–7μs). This result is due to Yampi's early development stage. Among other implementation problems, this first version uses memory copy operations that seriously decrease the overall performance. This has also impact in the maximal obtained bandwidth (around 52MB/s).

These preliminary results show that Yampi is at least able to compete in performance with the commercial message-passing packages. However, while the software is running stable in our configuration (4-nodes), it is in an early stage of development. Our experience with other SCI-related projects (Yasmin, DECK) make us believe that there is still room for improvement both in latency and bandwidth.

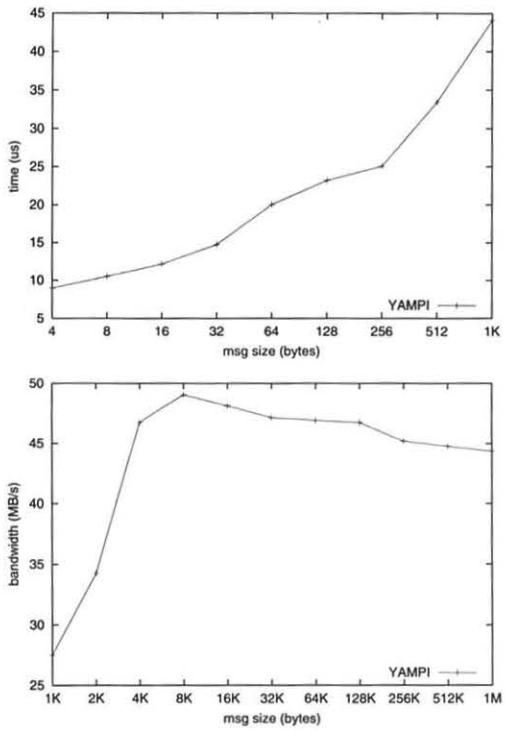

Fig. 6. Yampi performance results.

V. Concluding remarks

In this paper we presented our experiences in building clusters and developing tools and applications for the Scalable Coherent Interface. The results of several research fronts, including those of our own, are presented, what gives a very good picture of what has been done in the last decade for the SCI platform.

SCI was rather quietly adopted by several companies that recognised its superior concepts and protocols as well as its potential of high performance, but had no interest in implementing and providing SCI as an open interconnect. A number of proprietary implementations and products therefore appeared over the years, ranging from high-performance clusters interconnects, to shared-memory multiprocessor networks with cache coherence implemented in hardware, and high-speed I/O subsystem interconnects. In particular, the CC-NUMA (Cache Coherent Non-Uniform Memory Architecture) machines based on SCI technology from HP/Convex, Sequent, and Data General turned out to be quite successful.

Adoption of workstation clusters using SCI interconnect (and its DSM), is slower than expected, despite the superior performance characteristics of SCI cluster networks available today. The main reasons may well be that for many years there has been only one serious vendor of SCI adapters and switches, namely Dolphin Interconnect Solutions from Norway, and that a direct competing product in the segment of NORMA (No Remote Memory Access) clusters, the Myrinet from USA, had an enormous worldwide success driven by the broad adoption in the North American market (industries, government, military and universities).

We believe that the development of SCI and its influence is not finished yet. Some on-going experiences show that SCI MPI implementations in small clusters (from 8 to 32 pro-

cessors) are able to outperform Myrinet by over 10% for some applications. The SCI hardware implementation of a distributed shared memory allows the execution of shared memory applications in performance levels that are not possible with the DSM software emulation in Myrinet clusters. This allows the efficient execution of applications that are not easily ported with the message passing paradigm. It also allows much more choices for investigation, what is specially interesting for universities and research centres. Considering that both PCI interconnection cards cost the same (around US$1000 per node) the resulting SCI cluster is cheaper because no switch is needed up to 144 nodes (12×12 mesh). The Myrinet switch is especially expensive because it is implemented with a perfect crossbar—US$6000 for a 16 port stackable switch.

Based on our experience and the obtained results we are convinced that the Scalable Coherent Interface (SCI) is flexible, efficient, scalable, and it has an excellent cost-performance ratio, and therefore it should be seriously considered as an alternative for small- and mid-range cluster interconnection.

ACKNOWLEDGEMENTS

This work has been partially supported by grants from CAPES, DAAD and CNPq.

REFERENCES

[ÁVI 99] ÁVILA, R. B. et al. Modelagem e avaliação de desempenho de agregados conectados por tecnologia SCI. In: SBAC, 11., 1999, Natal, RN. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 1999. p.107–112.

[ÁVI 00] ÁVILA, R. B. et al. OptiSCI: a visual environment to optimize the placement of shared memory segments on a SCI cluster. In: SBAC, 12., 2000, São Pedro, SP. **Proceedings...** São Carlos: UFSCAR, 2000. p.129–135.

[BAR 00] BARRETO, M.; ÁVILA, R.; NAVAUX, P. The MultiCluster model to the integrated use of multiple workstation clusters. In: PC-NOW, 3., 2000, Cancun. **Proceedings...** Berlin: Springer, 2000. p.71–80. (Lecture Notes in Computer Science, v.1800).

[BAR 00a] BARRETO, M. E. **DECK**: um ambiente para programação paralela em agregados de multiprocessadores. Porto Alegre: PPGC da UFRGS, 2000. Dissertação de Mestrado.

[BAR 00b] BARRETO, M. et al. Implementation of the DECK environment with BIP. In: MYRINET USER GROUP CONFERENCE, 1., 2000, Lyon, France. **Proceedings...** Lyon: INRIA Rocquencourt, 2000. p.82–88.

[BOD 95] BODEN, N. et al. Myrinet: a gigabit-per-second local-area network. **IEEE Micro**, Los Alamitos, v.15, n.1, p.29–36, Feb. 1995.

[DOL 00] DOLPHIN interconnect solutions web. Disponível por WWW em http://www.dolphinics.no (abr. 2000).

[DOR 99] DORMANNS, M.; SCHOLTYSSIK, K.; BEMMERL, T. A shared-memory interface for SCI clusters. In: HELLWAGNER, H.; REINEFELD, A. (Eds.). **SCI**: Scalable Coherent Interface: architecture and software for high-performance compute clusters. Berlin: Springer, 1999. p.490. (Lecture Notes in Computer Science, v.1734).

[EIC 92] EICKEN, T. von et al. Active messages: a mechanism for integrated communication and computation. In: ISCA, 19., 1992, Gold Coast, Australia. **Proceedings...** New York: ACM, 1992. p.256–266.

[FIS 99] FISCHER, M.; REINEFELD, A. PVM for SCI clusters. In: HELLWAGNER, H.; REINEFELD, A. (Eds.). **SCI**: Scalable Coherent Interface: architecture and software for high-performance compute clusters. Berlin: Springer, 1999. p.239–248. (Lecture Notes in Computer Science, v.1734).

[GIA 98] GIACOMINI, F. et al. **Low-level SCI software requirements, analysis and predesign**. [S.l.]: ESPRIT Project 23174 — Software Infrastructure for SCI (SISCI), 1998.

[GM 99] GM. Disponível por WWW em http://www.myri.com/GM (dez. 1999).

[HEL 99] HELLWAGNER, H.; REINEFELD, A. (Eds.). **SCI**: Scalable Coherent Interface: architecture and software for high-performance compute clusters. Berlin: Springer, 1999. 490p. (Lecture Notes in Computer Science, v.1734).

[HER 98] HERLAND, B. G.; EBERL, M.; HELLWAGNER, H. A common messaging layer for MPI and PVM over SCI. In: HPCN, 1998, Amsterdam. **Proceedings...** Berlin: Springer-Verlag, 1998. p.576–587. (Lecture Notes in Computer Science, v.1401).

[HUS 99] HUSE, L. P. et al. ScaMPI—design and implementation. In: HELLWAGNER, H.; REINEFELD, A. (Eds.). **SCI**: Scalable Coherent Interface: architecture and software for high-performance compute clusters. Berlin: Springer, 1999. p.249–261. (Lecture Notes in Computer Science, v.1734).

[IBE 96] IBEL, M. et al. Implementing Active Messages and Split-C for SCI clusters and some architectural implications. In: SCIZZL, 6., 1996, Santa Clara, USA. **Proceedings...** [S.l.: s.n.], 1996.

[IEE 92] INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **IEEE standard for scalable coherent interface (SCI)**. New York: [s.n.], 1992. IEEE 1596-1992.

[MPI 94] MPI FORUM. **The MPI message passing interface standard**. Knoxville: University of Tennessee, 1994.

[OLI 01] OLIVEIRA, F. A. D. de. **Uma biblioteca para programação paralela por troca de mensagens de clusters baseados na tecnologia SCI**. Porto Alegre: PPGC da UFRGS, 2001. Dissertação de Mestrado.

[PRY 98] PRYLLI, L.; TOURANCHEAU, B. BIP: a new protocol designed for high performance networking on Myrinet. In: PC-NOW, 1., 1998. **Proceedings...** Berlim: Springer, 1998. p.472–485. (Lecture Notes in Computer Science, v.1388).

[SCH 99] SCHULZ, M. SCI-VM: a flexible base for transparent shared memory programming models on clusters of PCs. In: HIPS, 1999, San Juan, Puerto Rico. **Proceedings...** Berlin: Springer, 1999. (Lecture Notes in Computer Science, v.1586).

[TAŞ 98] TAŞKIN, H.; BUTENUTH, R.; HEISS, H.-U. SCI for TCP/IP for Linux. In: SCI-EUROPE, 1998, Bordeaux, France. **Proceedings...** [S.l.: s.n.], 1998.

[TAŞ 98a] TAŞKIN, H. **Synchronisationsoperationen für gemeinsamen speicher in SCI-clustern**. Paderborn: Universität GH Paderborn, 1998. Diplomarbeit.

[WOR 00] WORRINGEN, J. SCI-MPICH: the second generation. In: SCI EUROPE, 3., 2000, Munich, Germany. **Proceedings...** [S.l.: s.n.], 2000. p.10–20. Organizado como *conference stream* do Euro-Par'2000.

[ZOR 99] ZORAJA, I.; HELLWAGNER, H.; SUNDERAM, V. SCIPVM: parallel distributed computing on SCI workstation clusters. **Concurrency: Practice and Experience**, v.11, n.13, p.121–138, Mar. 1999.