Performance Potential of an SCI Workstation Cluster for Grid-Based Scientific Codes

Marcus Dormanns, Walter Sprangers, Hubert Ertl, Thomas Bemmerl
Lehrstuhl für Betriebssysteme, RWTH Aachen
Kopernikusstr. 16, D-52056 Aachen, Germany
Email: contact@lfbs.rwth-aachen.de

Abstract. *The recently emerging trend in parallel computer architecture is directed towards logically shared but physically distributed memory machines. Besides complete systems, it is possible to cluster off-the-shelf workstations with a suitable network to build such machines. This article focuses on performance considerations of such a cluster and their impact on grid-based code parallelization. We describe the capabilities of the interconnect and provide basic shared memory performance figures for different usage modes. We then discuss how application code performance is influenced by these characteristics. We concentrate on matrix-vector multiplication for irregular sparse matrices and show how to obtain satisfactory performance for different algorithmic / data-access patterns.*

1 Introduction

Workstations, clustered with the IEEE-standardized SCI (Scalable Coherent Interface; IEEE.1596) interconnect [9], offer an exciting new perspective for low-cost parallel scientific computing. While the standardization took already place in 1992, it's only recently that commercial products become available or are announced. Plug-in adapter cards for Sun workstations [4] are already available and those for PCs will be in 1997. The HP/Convex Exemplar SPP [1], was the first entire system. Clustered off-the-shelf Intel PentiumPro based multiprocessors from Sequent [10] and Data General [3], which will additionally be sold by other companies as an OEM version, are announced for 1997.

SCI firstly defines the physical link and logical protocol layer of a high-speed network (based on Gigabit and Gigabyte signalling technology) which enables message-passing with latencies of just a few microseconds and tens of MByte/s sustained throughput. Additionally, SCI contains a memory coherence protocol layer. This is the most exciting feature, since it provides physically distributed shared memory by virtual inclusion of memory segments of one machine from an SCI-cluster into another's virtual address space at user-level.

To achieve this, SCI directly maps memory-accesses targeting a remote memory address (which become visible on the system bus) into SCI transactions, which perform the requested operation on the remote memory without involving the remote processor. This enables all communication between systems to be done transparently by ordinary read and write operations from the user's point of view. Although, this is completely implemented in hardware, it does not come for free. Remote memory accesses are about one order of magnitude more expensive than local ones. The resulting distributed shared memory parallel machine model is called NUMA (Non-Uniform Memory Access), accounting for this architectural property. Employing this technology, it is now possible to extend the shared data parallelization programming model, which is the broadly accepted model of choice on SMP (Symmetrical MultiProcessing) machines, beyond a single SMP to whole clusters of workstations/PCs and most effectively clusters of SMPs.

It should be noted that SCI is not the only technology, aiming at providing hardware-based NUMA distributed shared memory. DEC's Memory Channel offers similar capabilities to SCI, but employing Encore's reflective memory as it's technological basis [8]. SGI's Origin is based on technology of the former Stanford DASH project [19]. Also, there are a lot of ongoing research project in this area, some of which are summarized in [16].

In section 2, we briefly discuss the capabilities of our SCI-coupled multiprocessor workstation cluster as far as they result from the specific implementation of the employed adapter cards and provide some performance figures in section 3. Both is done with special emphasis on aspects that influence application parallelization. In section 4, we study the performance potential of grid-based codes on such a platform. We end with some conclusions and remarks how forthcoming SCI-adapter cards might change the situation in section 5.

2 The Cluster Platform

At this time, our small development cluster comprises out of two Sun SS20 multiprocessor workstations equipped with two 50 MHz SuperSparc processors, capable of delivering 50 MFlop/s each. The SCI adapter cards from Dolphin Interconnect Solutions [4] come already in their second generation with an enhanced throughput and are simply plugged into the workstations' SBus (see figure 1a). Via their device driver, they are capable of:

- delivering low-latency, high-throughput message-passing and
- mapping memory segments of processes from each node (i.e. a workstation) into virtual address spaces of all other processes at all nodes (see figure 1b).

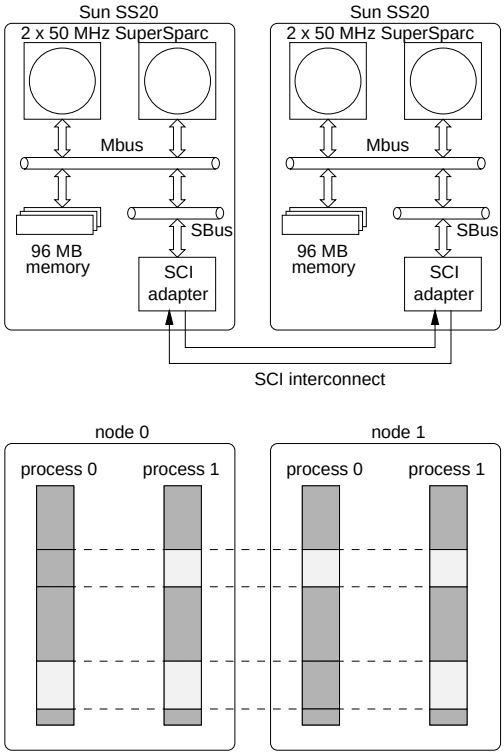

Figure 1: (a - top) Hardware configuration. (b - bottom) A possible logical configuration of two nodes with the address-spaces of two processes each. The dashed lines mark segments which are physically located on one node (dark) and virtually included via the SCI interconnect in other processes (light).

A serious drawback of this solution is that the adapters are plugged into the I/O-bus and not into the processors' bus itself. It is typically not possible to interrogate into the cache-coherence protocol within the SMP from an I/O-bus though it takes place on the processor bus. This limits the fraction of the SCI standard which is supported, the performance and finally influences the parallelization of applications:

- the SCI cache-coherency protocol is not implemented at all
- to hide complexity from the application programmer caused by the missing cache coherency protocol, the current device driver (version 1.5c.5) disables caching of remote memory segments (however, caching of globally shared memory segments at the owning node is not affected by this)
- though caching is disabled for remote memory accesses, remote memory fetches are not performed on the granularity of an entire cache-line; just the accessed data is fetched (e.g. 4 byte for an integer variable)

access mode	read latency	write latency
individual access	4.7 µs	2.2 µs
two processes from one machine concurrently	9.1 µs	4.3 µs
one process from each machine	5.6 µs	3.4 µs

Table 1: Remote read and write access latencies for different modes of operation.

Despite those limitations, the SCI interconnect enables to build clusters of standard workstations (and also PCs in the future) with a (partial) common address space which is not possible with other standard interconnects.

3 Basic Performance Characteristics

While most studies so far, e.g. [7, 14, 15], concentrate on SCI's message passing performance, we are mostly interested in it's shared memory capabilities for parallel processing.

First of all, the remote memory access latencies for a single datum are of interest. We discuss this for 64-bit floating-point numbers, which should be of most common interest. The results are summarized in table 1. The latency of a single remote read access is about 16 times, higher than a local cache miss that is satisfied by local main memory, which takes 284ns [12]. A write access, which does not require as many SCI transactions as a read access, takes about the halve.

Of special interest is how these numbers change if multiple accesses have to be served at the same time. For multiprocessor nodes, two cases can be distinguished: multiple accesses from a single machine (from both processors of the SMP node) to remote memory of another node, competing for the SCI adapter, and accesses from both machines into the respectively remote memory of the counterpart machine. It turned out that the current implementation of the protocol engine of Dolphin's SCI adapters is not capable of dealing with multiple outstanding SCI transactions. Multiple remote memory accesses are serialized resulting nearly in twofold access latencies for the case of accesses from two processors within one SMP node. Concurrent accesses from different nodes into the counterpart remote memory are not such worse, but result also in an increase of the access latency: about 20% for a read access and as much as 55% for a write access, indicating some congestion at the adapters also for this situation.

Besides the raw latency for an individual datum, the performance of accessing larger portions of data at the same time is of interest. One can imagine two situations: a

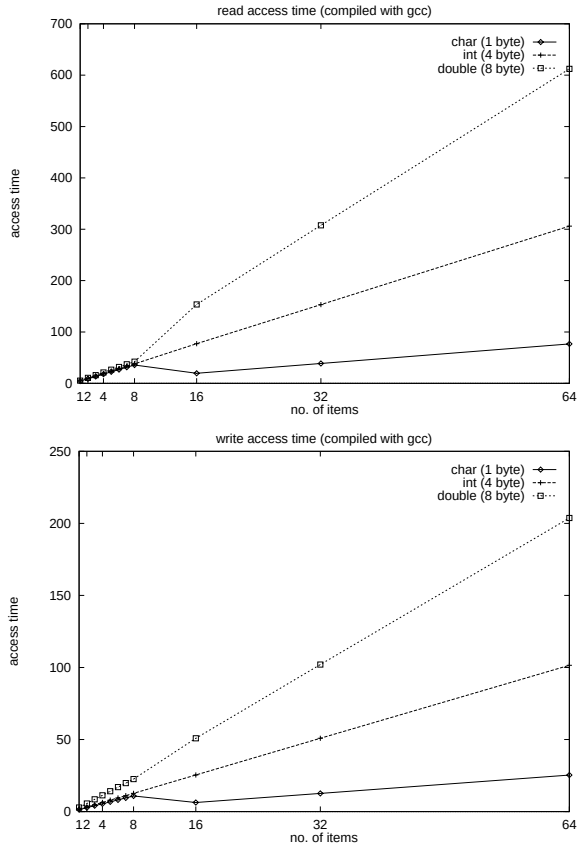

Figure 2: (a - top) Read and (b - bottom) write access time for structures that contain a varying number of characters (1 byte), integers (4 byte) and 64-bit floating point numbers (8 byte).

memory region is consecutively accessed by individual read/write operations or it is accessed as a whole, e.g. because it constitutes a larger data structure (e.g. velocities in three dimensions and pressure of a single grid point of a PDE in a CFD code, or thinking of a small submatrix for block-structured problems).

The fact that remote memory is not cached with the current implementation of Dolphin's SCI adapter cards, suggests to firstly copy the whole structure into a temporary local and therefore cacheable variable, before using it maybe multiple times. We found that the performance of accessing a whole remote data structure is dependent on the detailed compiler generated code, which depends on the data type and the number of items, comprising the structure. From this result different mappings of memory accesses to SCI transactions. The performance is depicted in figure 2.

In the case of read accesses (figure 2a), all read-operations of a structure containing up to 8 elements of one of the data type character (byte), integer (4 byte) and double-precision floating-point (8 byte) show nearly the same linearly increasing latency of about 4.7µs/item independent of the data type. This favors especially the access to double-precision floating-point numbers which refers to the largest amount of data per item. Above 8 items, the latency is exactly proportional to the amount of accessed data, not just to the number of items contained in a single structure. Similar artifacts occur for write operations as can be seen from figure 2b. This linear increase compared to the single access latencies of table 1 is the consequence of the disability to exploit the split-transactions, offered by the SCI-standard.

The resulting memory bandwidths amount to 1.52 MByte/s for a read access of up to 8 double-precision floating-point numbers collectively and 0.84 MByte/s for more items and different data types. Writes proceed with 2.52 MByte/s regardless of the specific data type (with an exception for structures of up to 8 characters). This has to be related to the 75 MByte/s local memory bandwidth of a Sun Sparc SS20 workstation, according to the *STREAM* benchmark [11].

4 Performance Potential of Irregular Grid Codes

Irregular grids are the underlying computational structure of many important problems and algorithms from computational science. Examples are iterative solvers of systems of linear equations from FEM and CFD problems and pair interaction calculation in molecular dynamics simulations. Exemplarily, we study the performance of matrix multiplication with an irregular sparse matrix, whose structure reflects the underlying problem's structure. Although very simple, it's data access pattern is representative for a large fraction of applications. For evaluation purposes, we employ 39 matrixes from the National Institute of Standards' MatrixMarket [13] which show a rich variety in structure (in detail: all matrices from the Harwell-Boing set [6] with a dimension greater then 3000 and some from the SPARSKIT set).

The following discussions are limited to a parallelism degree of 2. Definitely, this is not the final goal of parallel computing on an SCI cluster. But besides the fact that our small development cluster comprises just out of two machines, the focus of this article is not on scalability issues. Rather, hardware and problem characteristics will be discussed that have an impact on performance, as it was also done in [2] with synthetic problems. The matrices of the Harwell-Boing collection are mostly small ones and it is justified to assume that real parallelized problems are correspondingly larger but show similar structure.

The parallel implementation is based on *SMI*, the *Shared*

Memory Interface, which we are currently developing [5]. *SMI* is a parallelization library that provides comfort access to the capabilities of an SCI-cluster. Furthermore, it offers synchronization facilities, allows dynamic allocation of pieces of shared memory and switching between sharing and replication of data (that will turn out to be important below). Special emphasis is laid on accounting for the memory hierarchy: homogeneous memory performance inside each constituting multiprocessor building-block of the cluster and NUMA characteristic across machine boundaries.

4.1 Remote Reading

A first typical algorithm and data access pattern is characterized by each node of the underlying graph structure requiring data of all those nodes to that it is connected, to compute a result for it's own. In terms of linear algebra, this corresponds to a matrix-vector product $y = Ax$ that is performed row-by-row. Each element can be considered as a node, each non-zero element as an edge (it is not unusual that the structure of the graph of a matrix really corresponds to the structure of a grid, e.g. one from the discretization of a PDE). The natural (shared-memory) way to parallelize this is to divide the loop over the matrix-rows among the processors. The *SMI* library allows to store both vectors in globally shared memory segments, whose underlying physical memory is divided among the machines corresponding to the loop partitioning. While all write accesses to the solution vector are local, read accesses to the operand-vector are partially remote.

The point of reference for performance considerations is the time that is contributed to the total time of the matrix-vector product by a single non-zero element (one multiplication and one addition). To avoid modeling the performance of sparse matrix-vector multiplication that would be a challenge for it's own, we estimate it by measurement. For the chosen matrices, the matrix-vector multiplication is executed with 14% of the peak performance on average on a single processor. Therefore, the desired quantity is about 280ns on our 50 MFlops machine. This is dominated by memory access time, also in this strict local case. It should be noted, that this amount of work per edge, i.e. two floating-point operations, is quite small, compared to other grid-structures problems. Examples are molecular dynamics calculations that perform at the order of 100 floating-point operations per edge (neglecting all the others, e.g. integer operations) and block structured systems of linear equations.

In the parallelized case, the efficiency is decreased by non-local memory accesses. The performance can be related to problem-specific properties by looking at the fraction of non-zero matrix entries that cause a remote access to an element of the vector x, f.

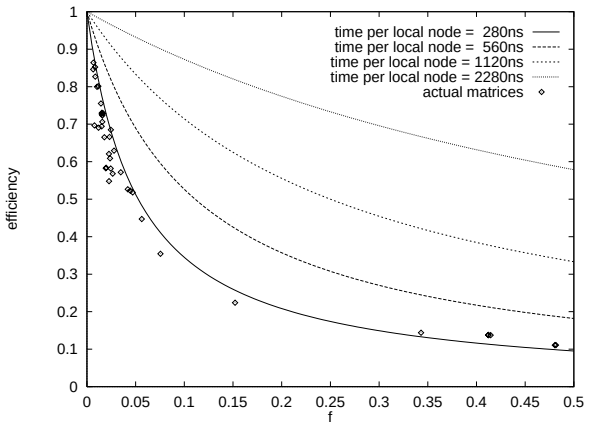

Figure 3: Efficiency of the brute-force implementation. Estimations for different compute-times per element and actual observed efficiencies.

The efficiency in the base parallel case without caching of remote data can be approximated by:

$$(1) \quad efficiency = \frac{(1-f)\cdot 280ns + f \cdot 5.6\mu s}{280ns}$$

For the employed matrixes with their concrete structure, the actual measured efficiencies together with this estimation are depicted in figure 3.

The resulting performance is not very satisfactory. But what can also be seen from figure 3 is that most of the problem structures exhibit a sufficiently amount of locality to show a much greater performance if the amount of work per edge of the underlying graph would be a somewhat more.

To assess how much of the overhead is induced by the absence of caching, we estimate the increased efficiency that would be observed with optimal caching. I.e. non of the vector elements that is reused several times is displaced from the cache during this time. The fraction of remote memory accesses would decrease by a factor of $1/r$, where r is the reuse rate of each remote element.

The resulting best-case efficiency results to:

$$(2) \quad eff = \frac{(1-f/r)\cdot 280ns + (f/r)\cdot 5.6\mu s}{280ns}$$

To deal with such problems, SMI allows to temporary replicate certain data structures at each computing node. Therefore the data is local and caching becomes possible. Furthermore, data can be replicated with the more efficient block transfers (see figure 2) than accessing each individual element. The analytical decrease of the parallel efficiency lack is proportional to the reuse rate. The actual observed benefits are depicted in figure 4.

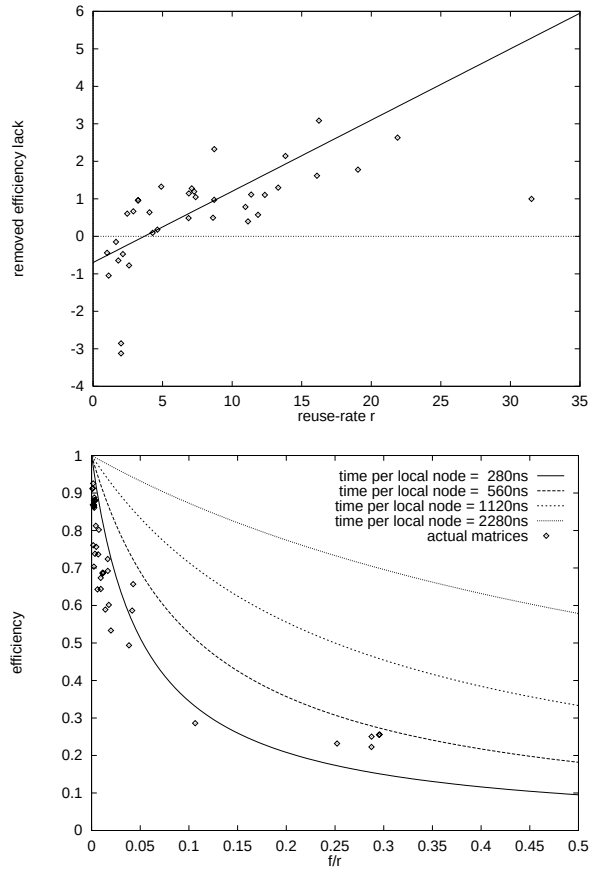

Figure 4: (a - top) Reuse rate within the matrices and the actual observed efficiency increase (b - bottom) Efficiency with explicite data replication.

Two reasons could be identified for the smaller efficiency lack decrease than estimated: first of all, the replication operation as implemented at this moment is not very efficient. This induces overhead especially for the larger data-sets. Secondly, the partitioning of the smaller problems shows some degree of load imbalance.

4.2 Remote Writing

The other common algorithm and data access pattern is that each node of the problem structure contributes to the results of all those nodes to that it is connected. In linear algebra, this corresponds either to a multiplication of the transpose of a given matrix with a vector row-by-row or to an ordinary matrix-vector multiplication if the matrix is stored in a row-major order. As before, the natural way to parallelize this is to partition the outer loop of the problem. With the before-mentioned physical memory distribution of the global shared vectors, all read operations of elements x_i can be served locally, but some of the accumulation operations $y_j \leftarrow y_j + a_{ij}x_i$ will concern remote elements. Beside performance issues due to the remote accesses (a read followed by a write operations in

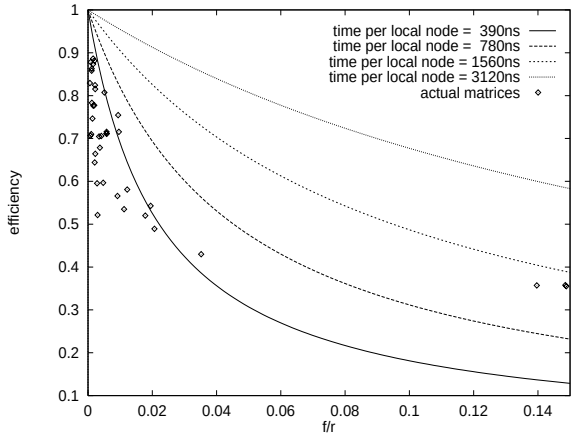

Figure 5: Performance of the remote write data access pattern.

this case), these accumulations have to be performed under mutual exclusion to guarantee correctness. The brute-force method of allocating a lock for each element that is acquired before and released after each such operation is prohibited due to it's overhead. Definitely, there might be tricky methods if the detailed sequence of accesses is known to allow locking on a very coarse granularity. But these are not generally applicable. At least for the matrix-vector multiplication, fine-grained locking on a NUMA-type machine (which takes about 15 µs per lock-acquire operation on our cluster) is prohibited.

Instead, we exploit *SMI*'s capability to temporary switch to a replication of the globally shared memory region for the vector y within each process. Then, each process can perform it's share completely local. Afterwards, SMI allows to combine all these local contributions to a single, globally shared final result. This can be performed very efficiently by exploiting locality and sparsity of accumulations for each process. I.e. for a grid-based problem which imposes locality that is also preserved in the problem partitioning (i.e. a row-partitioning of the matrix whose columns are permuted to reduce bandwidth) each process touches only a small fraction of elements of another partition. This matches the capability of the SCI-interconnect of fine-grained data transfer very well. The resulting parallel efficiencies are depicted in figure 5.

The sequential base time per non-zero element for this algorithms is 390ns. The efficiencies are quite satisfactory: about 68% on average but only about 1/3 of all problems with an efficiency below 70%.

5 Conclusions and Outlook

The central conclusion from this work is, that parallel programming on a shared-memory cluster of the type

that we considered, may result in terrible performance, if it's NUMA characteristic is ignored but programmed like a conventional SMP. At first glance, this might seem to be possible and attractive due to the great nominal performance figures of the SCI interconnect (its latency and throughput) in comparison to other cluster interconnects (e.g. fast Ethernet or ATM).

One of the reasons is the missing caching of remote data. While the cache coherency protocol layer is implemented in complete SCI-based systems, like those from Convex, Data General and Sequent, also forthcoming SCI adapter cards that allow the clustering of off-the-shelf workstations and PCs will probably not incorporate it. That's due to cost and difficulties, raised by the compulsions of the I/O-bus (see [18] for a discussion).

But just caching is not the universal remedy for all those problems. In the case of frequently write-shared data (section 4.2), cache invalidation by the cache coherency protocol prevents all advantages from caching. With the *Shared Memory Interface, SMI*, we are developing a parallelization support instrument, that allows to employ the highly attractive shared data parallelization paradigm also on NUMA clusters. It's capability to temporary switch to a replication of memory regions and to possibly combine the individual copies to a consistent shared view eventually, is one way to allow caching nevertheless. However, there is a significant potential to improve the performance of these functions. An alternative that we will consider in the future is to allow caching although the SCI cache-coherency protocol is not implemented. Doing so, a parallelization library like *SMI* has to support the programmer with cache-flushing in a comfort way.

Forthcoming SCI adapters for PCs will show a slightly reduced latency and an order of magnitude larger shared memory bandwidth due to multiple outstanding SCI transactions and streaming buffers [17]. Together with a sophisticated software support, SCI-clusters are an interesting platform for parallel processing.

To demonstrate the justification of this approach, we plan to parallelize the commercial relevant molecular dynamics code GROMOS. The interactions of adjacent atoms within a molecular ensemble form an irregular grid with a comparable large degree of connectivity. Exploiting the symmetry of pair-interactions results in a combination of the discussed remote read and remote write data access pattern.

References

[1] Abandah, G. A.; Davidson, E. S.: *Characterizing Shared Memory and Communication Performance: A Case Study of the Convex SPP-1000.* Technical Report CSE-TR-277-96, Dept. of EECS, Univ. of Michigan, Ann Arbor, 1996.

[2] Boyd, E. L.; Wellmann, J.-D.; Abraham, S. G.; Davidson, E. S.: *Evaluating the Communication Performance of MPPs Using Synthetic Sparse Matrix Multiplication Workloads.* Proc. of the Int. Conf. on Supercomputing, pp. 240-250, 1993.

[3] Clark, R.; Alnes, K.: *SCI Interconnect Chipset and Adapter: Building Large Scale Enterprise Servers with Pentium Pro SHV Nodes.* Proc. Hot Interconnects IV, 1996.

[4] Dolphin Interconnect Solutions, AS: *SPARC SBus-SCI Cluster Adapter Card.* White Paper, June 1995.

[5] Dormanns, M. Sprangers, W.; Ertl, H.; Bemmerl, T.: *A Programming Interface for NUMA Shared-Memory Clusters.* Submitted for publication.

[6] Duff, I. S.; Grimes, R. G.; Lewis, J. G.: Sparse *matrix test problems.* ACM Trans. Math. Soft., 15:1-14, 1989.

[7] George, A.; Todd, R.; Phillips, W.; Miars, M.; Rosen, W.: *Parallel Processing Experiments on an SCI-based Workstation Cluster.* Proc. 5th Int. Workshop on SCI-based High-Perf. Low-Cost Computing, pp. 29-39, March 1996.

[8] Gillet, R. B.: *Memory Channel Network for PCI.* IEEE Micro, pp. 12-18, Feb. 1996.

[9] Gustavson, D. B.: *The Scalable Coherent Interface and Related Standard Projects.* IEEE Micro, Feb. 1992, pp. 10-21.

[10] Lovett, T.; Clapp, R.: *STiNG: A CC-NUMA computer system for the commercial marketplace.* Proc. 23rd Annual Int. Symp. on Comp. Architecture, 1996.

[11] McCalpin, J. D.: *A Survey of Memory Bandwidth and Machine Balance in Current High Performance Computers.* IEEE TCCA Newsletter, Dec. 1995.

[12] McVoy, L.; Staelin, C.: *lmbench: Protable tools for performance analysis.* Proc. Usenix Annual Technical Conf., 1996.

[13] National Institute of Standards: *The Matrix Market*. http://math.nist.gov/MatrixMarket/index.html

[14] Omang, K.: *Performance results from SALMON, a cluster of Workstations Connected by SCI.* Research Report No. 208, Univ. of Oslo, Dept. of Informatics, Nov. 1995.

[15] Omang, K.; Parady, B.: *Performance of Low-Cost UltraSparc Multiprocessors connected by SCI.* Research Report No. 219, Univ. of Oslo, Dept. of Comp. Science, June 1996.

[16] Protic, J.; Tomasevic, M.; Milutinovic, V.: *Distributed Shared Memory: Concepts and Systems.* IEEE Par. & Distr. Technology, Vol. 4, No. 2, pp. 63-79, 1996.

[17] Ryan, S. J.; Gjessing, S.; Liaaen, M.: *Cluster Communication using a PCI to SCI interface.* IASTED Eighth International Conference on Parallel and Distributed Computing and Systems, 1996.

[18] Safranek, R. J.: Considerations in Implementing a System Based on SCI. Proc. of the SCIzzl-4 Workshop, 1995.

[19] Silicon Graphics Inc.: *Technical Overview of the Origin Family.* http://www.sgi.com/Products/hardware/servers/technology/overview.html