The Performance of Cache Coherency in SCI-based Multiprocessors*

Roberto A Hexsel
Depto. de Informática, Universidade Federal do Paraná

Nigel P Topham
Dept. of Computer Science, University of Edinburgh

Abstract *The Scalable Coherent Interface (SCI) is an IEEE/ANSI standard that defines a hardware platform for scalable shared-memory multiprocessors. This paper contains a quantitative performance evaluation of SCI-connected multiprocessors that assesses both the communication and cache coherence subsystems. 1D, 2D and 3D tori with 16 and 64 nodes are investigated. For the architecture (100MHz Sparc, 2 levels of caches) and workload simulated, it was found that raw network bandwidth seen by a processing element is under 100Mbytes/s. The 3-D torus is 10-15% faster than the 2-D torus for programs that generate high levels of network traffic. Otherwise, the differences in performance between 2-D and 3-D tori are negligible.*

1 Introduction

One of the architectural difficulties involved in the construction of efficient logically-shared physically-distributed memory multiprocessors is the interconnection system between processors and memory. Such interconnect must provide a low-latency high-bandwidth path between processors and memory in addition to efficient support to the physically-distributed logically-shared memory model.

The Scalable Coherent Interface (SCI) is an ANSI/IEEE standard [8] for a coherent memory interface to as many as 65520 processing nodes and defines a network that, in theory, satisfies the above criteria. SCI defines a physical layer (cabling, clock frequencies), a logical communication protocol (unidirectional point-to-point links) and a cache coherent protocol (invalidation-based distributed directory). The performance of the communication layer of SCI has been thoroughly investigated, both analytically and by simulation [14, 15, 2] and, it does satisfy the high-bandwidth low-latency requirement.

Bogaerts *et. al.*, in [2] evaluate the performance of large SCI fabrics to be used in data gathering and preprocessing in CERN's Large Hadron Collider. The technological parameters used are the same as here and their results are in broad agreement with those in [14], *i.e.,* SCI provides a low-latency high-bandwidth interconnect. Kofuji *et. al.*, in [11], report on simulation experiments with medium-sized multiprocessors using the parameters of a current implementation of a subset of the SCI standard (125Mbytes/s per link). [12] contains simulation results on page migration in an SCI-based multiprocessor. [7] presents simulation results that relate cache coherency and interconnect traffic for SCI rings.

SCI provides a simple yet efficient synchronization mechanism called *Queue on Lock Bit* (QOLB). Its implementations and performance are compared to other synchronization

*VIII Simp Bras de Arquit de Computadores – Proc de Alto Desempenho, pp.47-56, Recife, Aug 1996.

mechanisms in [18, 1]. As they report, QOLB fits quite naturally with SCI's distributed sharing-lists. Kägi *et. al.* present simulation results of an architecture and workload similar to those discussed here [10]. They study the interactions of techniques to reduce the overheads associated to physically distributed memory. The techniques investigated are synchronization with QOLB, weaker memory consistency and cache coherence protocol optimizations for pair-wise sharing, all of which are supported by SCI.

The performance of shared-memory multiprocessors depends on a fast network as well as on efficient implementation of the shared-memory abstraction. One technique widely employed in implementing shared-memory is to use a cache coherence protocol to keep the physically distributed memory in a coherent state. This paper investigates the interactions between cache coherence and network latencies on an SCI-based multiprocessor executing real programs. Some of the questions we attempt to answer are "what is the cost of sharing data using SCI's distributed directory protocol under differing patterns of sharing?" and, "how efficiently does coherency related traffic uses SCI's high bandwidth network?". Answers to these questions can be found by looking at the behavior of a machine, with processors and a memory hierarchy, executing real code, since in such a system all of SCI's components interact in a realistic manner.

This paper contains a performance evaluation of a family of multiprocessors based on SCI interconnects where the influence of the cache coherence protocol on performance is investigated. The experiments also relate the performance of the memory hierarchy to that of the interconnect. The topologies studied are 1D (rings), 2D (meshes) and 3D tori (cubes) with 16 and 64 processors. The architecture simulator is driven with address sequences generated as a by-product of the execution of real programs. The workload consists of two programs from the SPLASH suite [16] (MP3D and Water) and three parallel loops (Gaussian elimination, matrix multiplication and all-to-all minimum cost paths). The simulation environment and the workload used to drive the architecture simulator are described in Section 2. Section 3 discusses the overheads imposed by the cache coherency protocol. Section 4 discusses the network performance of SCI 1D, 2D and 3D tori. Section 5 presents our conclusions.

2 Simulation Environment

The architecture simulator consists of an approximate model of the SCI link interfaces and of a detailed model of the distributed cache coherence protocol. The model of the ring interfaces is described in [6]. The traffic conditions are measured at $10\mu s$ intervals and network delays are estimated from these measurements. The model of the cache coherence protocol mimics the "typical set coherence protocol" as defined in [8]. The simulator consists of two Unix processes: the *memory reference stream generator* and the *architecture simulator*. The reference stream is piped to the architecture simulator which computes the latency of each (simulated processor) reference to memory. This latency is used by the reference stream generator to choose the next simulated thread to run. The address sequences used to drive the simulator are generated by instrumenting parallel programs with Symbolic Parallel Abstract Execution (SPAE) [5]. SPAE is based on the GNU `gcc` compiler and allows for tracing parallel programs at any desired level of detail. Typically, a simulation run takes from 1 to 100 CPU hours on a lightly loaded Sparcstation2. In order to simplify the simulator, it is assumed that on data accesses the concurrent instruction fetch hits in the primary cache and, accesses to local data and instructions do not cause any traffic on the ring. It is also assumed that page faults have zero cost.

The Simulated Multiprocessor The multiprocessor consists of a number of processing nodes interconnected by SCI links. The CPU is a 32-bit 100MHz SPARC processor that performs an instruction fetch and possibly a data read/write access on every clock cycle. The simulated processors always stall on memory references (both read and write), thus the memory model is sequential consistency [13]. The memory hierarchy comprises three levels: split primary caches, unified secondary cache and main memory. The primary caches are 8 Kbytes each, direct mapped. The data cache is write-through with no block allocation on write misses. The secondary cache is direct mapped and, for private data references it is copy-back with no block allocation. The secondary cache size is 256 Kbytes. On all three levels of the memory hierarchy, cache and memory lines (blocks) are 64 bytes, as per the SCI standard. The memory hierarchy satisfies the multilevel inclusion property and the SCI coherency protocol actions affect only the secondary caches. The access latency for the secondary caches is 3 processor cycles. Loading a line from the secondary cache into the primary caches or SCI controller costs 3 processor cycles plus 2ns per 64 bit word (16ns). Loading a line from/to memory costs 120ns of access latency plus 10ns per 64 bit word (80ns).

The 1-, 2- and 3D k-ary n-cube networks are implemented by having one or more pairs of SCI links on each node, with each pair belonging to a different ring. Each ring in the network is modeled independently. The cost of switching dimension is five extra network cycles (10ns). The cost of a transaction is computed by adding up the memory and network delays on all rings in the path from requester to responder. The router employed in the simulator is based on the *e-router* [3]. The e-router is shown in [9] to be deadlock-free on SCI-based k-ary n-cubes. The path from source to destination is always chosen by inserting the packet at the highest dimension, where it travels as far as possible before being switched onto the next lower dimension. Deadlock avoidance is ensured by the partitioning of network queues into a set of ordered classes, with the queues in each dimension comprising each of the classes.

The Workload The workload used to investigate the behavior of SCI multiprocessors consists of three parallel loops and two real programs. The parallel loops, based on `doall` loops are small and exhibit a well defined pattern of memory references. The real programs are much larger and are part of the SPLASH suite [16]. A detailed description of `mp3d()` and `water()` can be found in [16]. `mp3d()` is simulated for 50 time steps and `water()` for 4 time steps. The arrays and variables that hold shared data are allocated to a specific range of addresses. The architecture simulator treats references to these addresses as references to shared data.

`ge()` solves a system of linear equations by Gaussian elimination and backwards substitution. It is assumed that the system of equations has some property that makes Gaussian elimination without pivoting numerically stable (*e.g.* diagonal dominance). The algorithm runs through several elimination stages. Each stage consists of a vector scale operation of the form $(x_{k+1} = cx_k)$ followed by a (rank-1) update of the matrix $(A_{k+1} = A_k + dxy)$ where x and y are vectors, c and d are scalars. At the k-th stage, matrix A has dimension $((n-k) \times (n-k+1))$. `mmult()` computes $C = A \times B$ for square matrices A and B. The algorithm consists of three nested loops and each processor computes a slice of the result matrix. `paths()` is a member of the class of transitive closure algorithms. For a graph with N nodes, `paths()` finds the lowest cost path from each node to every other node [4]. The vertices are labeled with the distance between the nodes they join and are stored in the matrix D. Thus, $D[i, j]$ is the distance between nodes i and j and, absence of a vertex is represented by infinite cost. The simulated graph is a random graph with out-degree 6. The three loops are $O(n^3)$ and input data-set sizes are scaled as $1.26 \times nodes$.

Scalability of data sets For a given program, an architecture is said to be *scalable under constant work per processor* if the execution time remains roughly constant as more processors are added *and* the data-set size is increased so that the work per processor remains constant. One way of ensuring a uniform distribution of work across processors is by keeping the number of references to shared data (roughly) constant. By choosing a large enough number of references, the caches can be fully and equally exercised, thus minimizing distortion caused by cold starts. Data-set sizes were chosen so that there are at least one million references to shared data. Detailed reference counts for the simulations reported here can be found in [6]. Table 1 shows, for the programs in the workload the number of molecules simulated (`mp3d()`, `water()`), the size of the matrices (`ge()`, `mmult()`) and the size of the graph (`paths()`).

program	data $\downarrow$ procs. $\rightarrow$	1	16	64
`mp3d()`	molecules (10^3)	3.0	15.2	34.2
`water()`	molecules	54	237	512
`ge()`	matrix rows	136	343	545
`mmult()`	matrix rows	100	252	400
`paths()`	graph vertices	70	176	280

Table 1: Data-set sizes for the workload.

3 Cache Coherency

This section compares the performance of the three topologies and relates performance to sharing behavior and cache coherence activity. Since the simulated machine sizes are not all the same on the three topologies, comparisons are drawn for same-sized multiprocessors. 1D tori were simulated with 16 nodes, 2D with 16 (4x4) and 64 nodes (8x8) and, 3D tori with 64 nodes (4x4x4). Detailed statistics for hit, flush and purge ratios can be found in [6]. The *flush ratio* is the number of cache lines flushed for each reference to the secondary caches. The *purge ratio* is the number of sharing-lists purged per write to shared-data and, the *sharing-list length* is the number of copies in the sharing-list that have to be purged prior to updating the cached line.

Figure 1 compares the performance of the workload on 16- and 64-node tori. Execution time is split into six types of activity: (1) the time spent fetching and executing instructions, (2) time spent on references to private data, (3) time spent on references to node local shared-data, (4) time spent on references to remote shared-data, (5) time spent on network delays (netwrk) and, (6) time spent waiting at barriers and locks (syncr). Time spent on references to remote data include the cost of purging sharing lists. References to instructions and private data that miss in the secondary cache may cause the flushing of a shared line; otherwise, these references do not cause any network activity. References to node local shared-data that miss in the secondary cache may cause a line to be flushed to local or remote memory; the former does not cause network traffic.

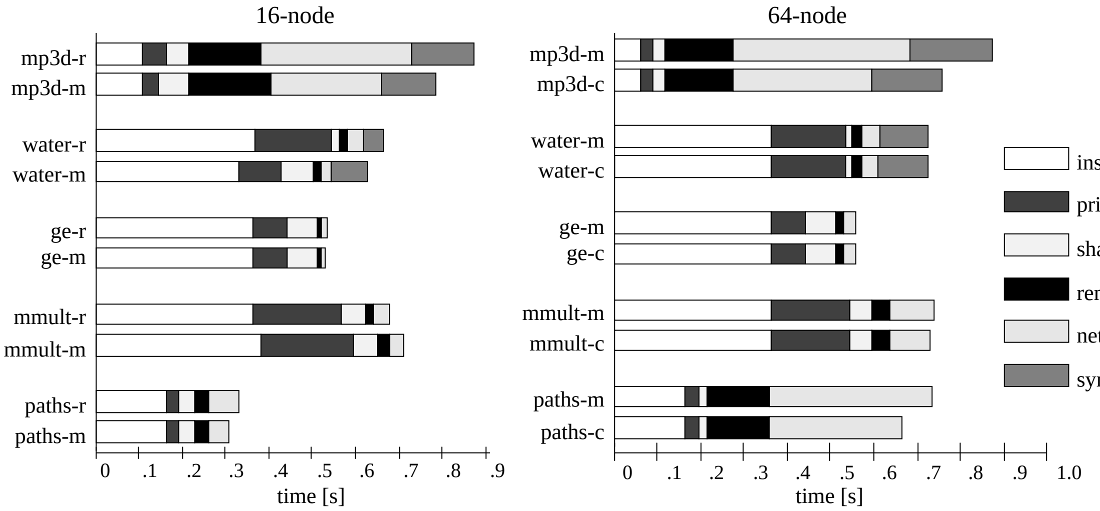

Figure 1: Performance of 16- and 64-node multiprocessors. The suffixes -r, -m and -c stand for ring, mesh and cube, respectively. Data sets scale up with machine size.

As can be seen in Figure 1, the performance of `mp3d()` is limited by network delays and references to shared data. Molecule data in this program is migratory [17]. At any simulation step, data for a given molecule are mostly used by one processor but as simulation progresses those data migrate from processor to processor. At the beginning of each simulation step, many cache lines are flushed and replaced and this involves 1 or 2 coherency protocol transactions to flush the old line plus 1 or more to fetch the new line. Towards the end of the steps, as data are updated, stale read-shared copies have to be invalidated and sharing-lists purged, with 2 or more transactions if copies exist. On a 16-node ring, every reference to the secondary cache causes an average 0.41 lines to be flushed and, every shared-data write causes an average 0.99 sharing-lists to be purged by invalidating 1.2 copies. On 16-node meshes, `mp3d()` displays virtually the same behavior (the above figures are 0.40, 0.99 and 1.2 respectively) but the higher network capacity yields a performance gain of 11%. In some of the workload, the variation in the time spent referencing private and shared data stems from the changes in the mapping of pages to nodes.

`mp3d()` displays the same type of behavior on 64-node systems as in 16-node systems but, because of the larger data sets, the shared-data hit ratios are lower (< 0.58). Thus, processors spend most of the time waiting for the completion of remote shared-data references. As before, the larger network capacity of the 3D torus yields a performance gain of 15%. `water()` `ge()` and `mmult()` do not make many references to remote shared-data and do not use much interconnect bandwidth. Hence they show only slight improvements in performance in the higher dimensionality network. When comparing to 16-node systems, these programs spend more time on references to shared-data because the data sets scale up with machine size, hence the hit ratios decrease because of the higher number of capacity, compulsory and coherency misses.

The algorithm used in `paths()` is not well suited to physically distributed memory because the processors scan the entire graph matrix when computing the minimum cost paths. This causes the setup of long sharing-lists which have to be purged on updates to the cost of paths. In the mesh, the shared-data read hit ratios at secondary caches is 0.65. Each write to shared-data purges an average 0.995 sharing-lists, invalidating an average 9.3 copies per purge. Besides the high levels of write-sharing, there are many cache-line mapping conflicts since each reference to secondary caches causes an average 0.90 lines to be flushed. For the cube, the above figures are: hit ratio 0.65; 0.994 sharing-lists purged with 10.4 copies each; 0.89 lines flushed per reference. Because of the increase in data set size, on 64-node machines, `paths()` spends a large fraction of the time waiting for the completion of shared-data references.

The cost of cache coherency In order to assess the overheads imposed by the SCI cache coherence protocol and interconnection network, simulations were run on an ideal shared-memory multiprocessor. The IDEAL multiprocessor has a network with zero propagation delay and a coherence protocol whose actions have zero latency. When a cache line is flushed or purged, the coherence protocol actions take effect instantaneously. The results for `mp3d()`, `water()` and `paths()` are shown in Figure 2. For each of the three programs, the results for IDEAL are shown above those for the SCI-based machine. In IDEAL, the segments labeled **shared** account for references to shared data in local as well as remote memory. References to local or remote shared-data have the same cost.

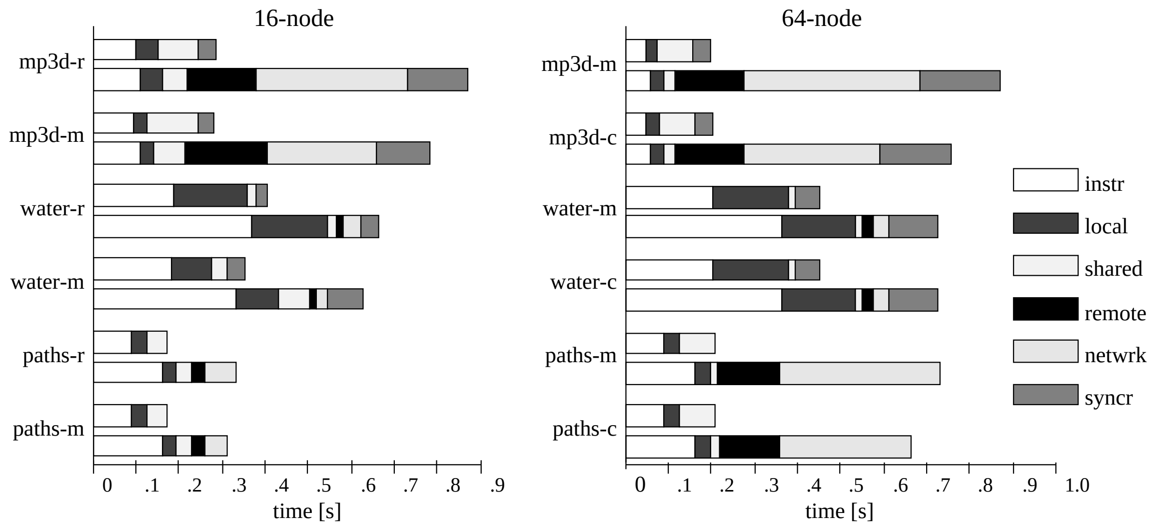

Figure 2: Performance of IDEAL *versus* SCI multiprocessor. For each program, the ideal performance is shown above the simulated performance. The suffixes -r, -m and -c stand for ring, mesh and cube respectively.

In IDEAL, there is a good degree of overlapping between executing an instruction and a data reference in the same processor clock cycle (p-cycle). The references that miss in either of the caches incur in the cost of a memory access (13 p-cycles). Because the SCI-based multiprocessor lacks any latency hiding devices, the latencies of each and every cache miss are fully exposed *e.g.* references to remote data cost at least two round-trip delays ($2 \times$ 10-20 p-cycles) in addition to cache and memory accesses. For the programs shown in Figure 2, the combination of miss, flush and purge ratios makes the fraction of time spent on shared-data references be about double that in IDEAL. This doubling accounts for the protocol actions involved in flushing stale data, purging sharing-lists and fetching fresh data.

4 Bandwidth and Latency

In the SCI-based multiprocessor studied here, programs that endure low hit ratios and/or have much write-sharing will be slowed down by network congestion. Read misses may cause the flushing of shared lines; write misses may cause both a flush and the purging of the sharing-list in addition to fetching the missed line. Each of these events might trigger one or more protocol transactions across the network and each transaction consists of one request plus its echo and one response plus its echo to cross the network. For the workload simulated, the percentage of packets that carry 64 bytes of data is under 10-13%. One half of all packets are echoes (8 bytes/packet) and the remaining 37-40% packets carry cache coherency information such as list pointers and invalidation requests (16-32 bytes/packet).

The underlying transport mechanism in SCI is the asynchronous insertion ring. The transmission of a packet is completed when its echo is received by the transmitter. The time lapse between the insertion of a packet into the output buffer and the receipt of its echo is defined as the *round-trip delay* of the network. Latencies incurred in accessing memory and caches are not included. The static (no traffic) latency for a 16-node ring is 116ns, for an average packet size of 22 bytes.

Node throughput is the number of symbols inserted by the node per time unit and is a measure of the amount of coherence-related traffic generated by the processor and cache/memory controllers. Note that the measured throughput includes packet header overhead. Data-only throughput is about 20 to 30% of raw throughput. Given that under 14% of all packets injected into the network carry 64 bytes of data while all except echo packets carry cache coherency information, raw throughput is a better measure of overall system performance. In a mesh, a single processor request can generate up to two packets, one on each network dimension. The first packet is injected by the processor and the second by the SCI interface of the node where the change of dimension occurs. Similarly for cubes. Processor throughput is thus computed by taking only traffic generated by the on-board processor and cache/memory controller, and dividing it by the execution time.

The number of packets a node can transmit per time unit depends on the *traffic* on the network. The traffic seen by a node at its ring interface(s) is defined as the number of symbols per time unit that is output by the ring interface(s). It consists of the packets inserted by the node itself plus the packets passing through that node towards downstream nodes. Traffic levels around 600 Mbytes/s are a limiting factor in the performance of SCI-connected rings since, at these levels, network delays are holding down the rate of network requests by processors. Bypass buffers have utilisations of over 50% and that leaves few opportunities for injecting packets into the rings.

Throughput versus Latency A plot of throughput *versus* latency for a given network shows how well processors can use the available bandwidth. Figure 3 shows the plots for the three topologies discussed here. The data for each of the programs keep their relative position on all four topologies. Network saturation is evident from the slope of the line on the plot for 16-node ring. `mp3d()` causes the highest traffic and endures the longest delays. The traffic levels on the mesh are much lower because of its larger network capacity. Hence, `mp3d()` does not drive the 16-node mesh into saturation. However, on the 64-node mesh, this program starts to saturate the mesh. The higher capacity and smaller distances of the cube yield lower delays on the programs that cause high levels of traffic.

The lines of throughput *versus* latency for 16-node machines show a marked improvement in the 2D torus relative to the 1D torus. This stems from the increased network capacity and a lessening of contention for the insertion of messages into the rings. The same effect can be seen on the 64-node machines, since the data sets scale up with machine size, the effects of increasing network capacity are readily apparent on the 2D and 3D tori. With the data sets used in the simulations, there would be little to be gained from using higher dimensional networks (≥ 4) since the delays incurred in changing dimension would offset gains from increased capacity and smaller diameter.

In terms of overall performance, cubes are 10–15% faster than meshes with programs that generate high levels of network traffic, that is, can drive the network closer to saturation. For programs that produce low levels of traffic, the differences between meshes and cubes are negligible. One has to balance the additional cost of increasing the dimensionality of the network against the potential improvement in performance. On the 64-node systems, this means using 64 additional SCI interfaces if a 3D torus is employed – by adding 50% more link interfaces the speed increases by 15%. Considering the small increases in performance with the workload used here, the 2D torus is a better choice given the price-performance differences.

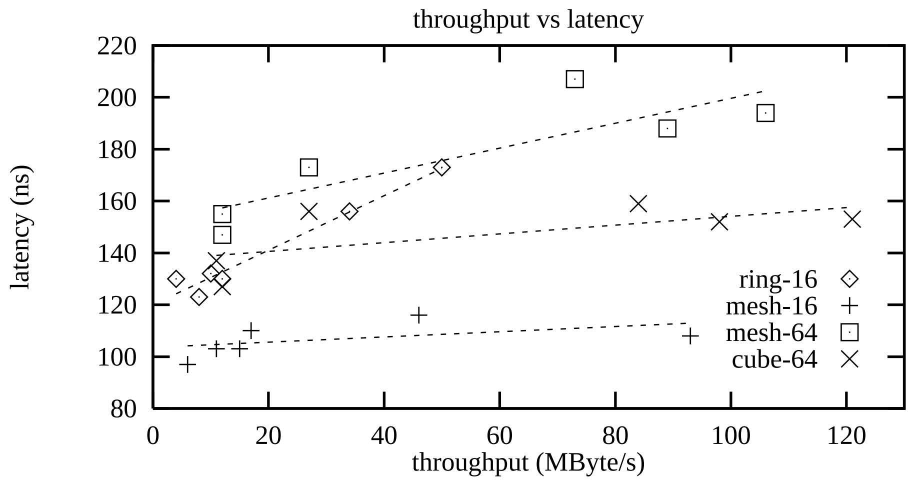

Figure 3: Plots of throughput versus latency for rings, meshes and cubes.

5 Conclusion

This paper presents a performance evaluation study of SCI-based shared memory multiprocessors. Previous studies of SCI-based systems have concentrated on network performance and to some extent ignored the influence of the cache coherence protocol. Here, the interactions between interconnection network and cache coherence protocol are investigated. A multiprocessor system was "implemented" in the simulator with components compatible with the current levels of performance. Two architectural parameters were investigated, namely machine size, and interconnection topology. Machines were simulated with sixteen and sixty-four 100MHz Sparc processors connected in 1D, 2D and 3D tori.

In order to assess the overheads imposed by the cache coherency protocol, the simulated multiprocessor was compared to an ideal machine with zero latency coherency protocol and interconnect. For the workload simulated, it was found that the cache coherence protocol increases the fraction of execution time spent on references to shared-data by between 40 and 160%. On top of that, network latencies should be considered as well. Latency tolerating mechanisms, write buffers as a minimum, should be implemented in the nodes. Otherwise, not unlike other distributed memory systems, performance would be rather poor.

Acknowledgments Roberto Hexsel was supported by a grant from CAPES. We thank Bruno Müller Jr for valuable comments on an earlier draft of the paper.

References

[1] N M Aboulenein, J R Goodman, S Gjessing, & P J Woest. Hardware support for synchronisation in the Scalable Coherent Interface (SCI). In *Proc of the 8th Intl Parallel Processing Symposium*, pages 141–150, Cancún, 1994. IEEE Comp Soc Press.

[2] J A C Bogaerts, R Divià, H Müller, & J F Renardy. SCI based data acquisition architectures. *IEEE Trans. on Nuclear Sciences*, 39(2), April 1992.

[3] William J Dally & Charles L Seitz. Deadlock-Free message routing in multiprocessor interconnection networks. *IEEE Trans. on Computers*, C-36(5):547–553, May 1987.

[4] N Deo, C Y Pang, & R E Lord. Two parallel algorithms for shortest path problems. Tech Report CS-80-059, Washington State Univ, March 1980.

[5] D Grunwald, G J Nutt, D Wagner, & B Zorn. A parallel execution evaluation testbed. Tech Report CU-CS-560-91, Dept of Computer Science, Univ of Colorado, November 1991.

[6] Roberto A Hexsel. *A Quantitative Performance Evaluation of SCI Memory Hierarchies.* PhD dissertation, Dept of Computer Science, Univ of Edinburgh, October 1994. Tech Report CST-112-94.

[7] Roberto A Hexsel & Nigel P Topham. The performance of SCI multiprocessor rings. *Journal of the Brazilian Computer Society*, 1(2):24–37, July 1995.

[8] IEEE. *IEEE Std 1596-1992 – Standard for Scalable Coherent Interface.* IEEE, 1992.

[9] Ross E Johnson & James R Goodman. Interconnect topologies with point-to-point rings. Tech Report 1058, Computer Sciences Dept, Univ of Wisconsin–Madison, December 1991.

[10] A Kägi, N Aboulenein, D C Burger, & J Goodman. An analysis of the interactions of overhead-reducing techniques for shared-memory multiprocessors. In *Proc of the Intl Conf on Supercomputing (ICS95)*, pages 11–20, Barcelona, July 1995. ACM Press.

[11] S T Kofuji, C A P da Silva, L G G Kiatake, M H S Cintra, & J A Zuffo. Anéis e hierarquias de anéis com interconexões ANSI/IEEE SCI. In *VII Simp Brasileiro de Arquit de Computadores – Proc de Alto Desempenho*, pages 11–25, Canela, RS, julho 1995.

[12] S T Kofuji, M X T Delgado, E D M Ordoñez, & J A Zuffo. Efeito da migração de páginas no SPADE-I: um multiprocessador de larga escala com memória compartilhada. In *XXII Semin Integrado de Software e Hardware*, pages 61–73, Canela, RS, julho 1995.

[13] Leslie Lamport. How to make a multiprocessor that correctly executes multiprocess programs. *IEEE Trans. on Computers*, C-28(9):690–691, September 1979.

[14] S L Scott, J R Goodman, & M K Vernon. Performance of the SCI ring. In *Proc. 19th Intl. Symp. on Computer Arch.*, pages 403–414. ACM Comp Arch News 20(2), May 1992.

[15] Steven L Scott & James R Goodman. The impact of pipelined channels on k-ary n-Cube networks. *IEEE Trans. on Parallel and Distributed Systems*, 5(1):2–16, January 1994.

[16] J P Singh, W-D Weber, & A Gupta. SPLASH: Stanford ParalleL Applications for SHared-memory. Technical Report CSL-TR-91-469, Computer Science Dept, Stanford Univ, April 1991. Also in ACM SIGARCH Comp Arch News 20(1).

[17] Wolf-Dietrich Weber & Anoop Gupta. Analysis of cache invalidation patterns in multiprocessors. In *3rd Intl. Conf. on Arch'l. Support for Progr. Lang. and Oper. Sys.*, pages 243–256. ACM Comp Arch News 17(2), April 1989.

[18] Philip J Woest & James R Goodman. An analysis of synchronization mechanisms in shared-memory multiprocessors. Tech Report 1005, Computer Sciences Dept, Univ of Wisconsin–Madison, April 1991.

Appendix – The Scalable Coherent Interface

The project that became SCI started as the design of a high performance bus. Early on it became obvious that the bandwidth of a bus would always place a hard limit on performance. The solution was to employ point to point unidirectional connections since these allow for much higher clock rates and richer interconnection patterns. SCI defines three subsystems, namely the physical-level interfaces, the packet-based logical communication protocol, and the distributed cache coherence protocol. The standard supports a general interconnect, providing a coherent shared-memory model, scalable up to 64K nodes. An SCI node can be a memory module, a processor-cache pair, an I/O module or any combination of these. The number of nodes on a ring can range from two to a few tens. For most applications, a multiprocessor will consist of several rings, connected together by SCI switches.

Physical Interfaces. The physical interfaces are high-speed uni-directional point-to-point links. One of the versions prescribes links 16-bit wide which can transfer data at peak speed of 1 Gbyte/s. A *symbol* consists of two bytes and every 2 ns a symbol is gated out of the output link.

Communication Protocol The communication protocol comprises the specification of the sizes and types of packets and the actions involved in the transference of information between nodes. A packet consists of an unbroken sequence of 16-bit symbols. It contains address, command/control and status information plus optional data and a check symbol. A command/control packet can be 8 or 16 symbols long, a data packet is 40 symbols long and an echo packet, 4 symbols. A data packet carries 64 bytes of data.

The protocol supports two types of action: *requests* and *responses*. A complete transaction, for example, a remote memory data-read, starts with the requester sending a request-send packet to the responder. The acceptance of the packet by the responder is acknowledged with a request-echo. When the responder has executed the command, it generates a response-send packet containing status information and possibly data. Upon receiving the response-send packet, the requester completes the transaction by returning a response-echo packet. The communication protocol ensures forward progress and contains deadlock and livelock avoidance mechanisms.

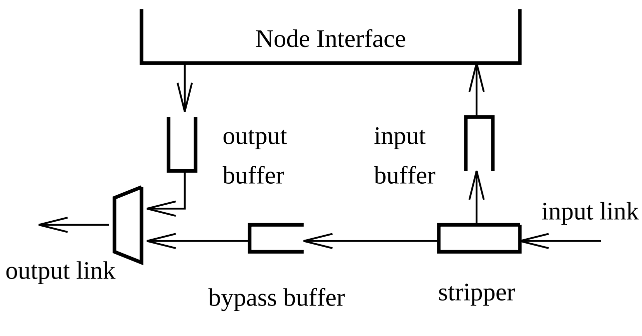

Figure 4: SCI link interface.

The network access mechanism used by SCI is the register insertion ring. Figure 4 shows a block diagram of the link interface. A node retains packets addressed to itself and forwards other packets to the output link. A request transaction starts with the sender node placing a request-send packet, addressed to the receiver node, in the output buffer. Transmission can start if there are no packets in the bypass buffer. At the receiver, the stripper parses the incoming packet and, if addressed to this node, diverts it to the input buffer. In this case, the stripper generates an echo packet addressed to the sender and inserts the echo in place of the 'stripped' packet. If there is space at the input buffer, the echo carries an ack (positive acknowledge) status. Otherwise, the packet is dropped and a nack (negative acknowledge) is returned to the sender who will then retransmit the request-send packet.

Cache Coherence Protocol The SCI cache coherence protocol is based on a write-invalidate distributed directory. Each cache line tag contains pointers to the next and previous nodes in the doubly-linked sharing-list. The address of a line consists of a 16-bit node-id and 48-bit offset. At memory, the state of a line with no copies is home. If there are read-only copies of a clean line, its state at memory is gone. An exclusive copy is dirty. Figure 5 shows two caches holding copies of a given line. Memory module M is the home of the line which is in state gone. The most recent copy is held by the head of the sharing-list (state head) and other copies are either at the mid or tail of the sharing-list. A line whose contents at the cache are invalid is marked invalid.

When a node requests a copy of a line to its home node, the memory controller either sends it directly (home) or, forwards the request to the head of the list who satisfies the request (gone, dirty). The requesting node then becomes the new head. When the contents of a line have to be purged to make room for other data, the flushing cache detaches itself from the sharing-list by sending messages to its neighbors in the list to update their forward and backward list-pointers.

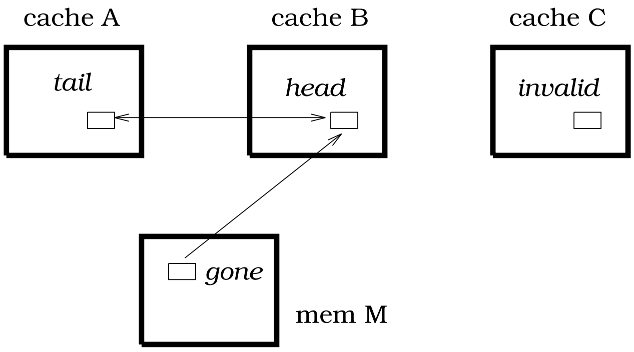

Figure 5: Sharing list pointers and cache/memory states.

Before writing to a shared line, the writing processor must become the head of the sharing-list and must purge the other entries in the list to obtain exclusive ownership of the line. Upon becoming the new head, the writing processor sends an invalidation message to the next cache in the list who returns its forward pointer. With that pointer, the writing processor can invalidate the following copy. This process is repeated until the sharing-list has collapsed to a single exclusive copy.