The Performance of SCI Multiprocessor Rings

Roberto A Hexsel
Depto. de Informática
UFPR – Centro Politécnico
Caixa Postal 19081
81531-970 Curitiba, PR
Fax +55 (0)41 267 4236
`roberto@inf.ufpr.br`

Nigel P Topham
Dept. of Computer Science
Edinburgh University
JCMB – The King's Buildings
Edinburgh EH9 3JZ
Scotland
`npt@dcs.ed.ac.uk`

Abstract

The Scalable Coherent Interface (SCI) is an IEEE standard that defines a hardware platform for scalable shared-memory multiprocessors. This paper contains a quantitative performance evaluation of an SCI-connected multiprocessor that assesses both the communication and cache coherence subsystems. For the architecture and workload simulated, it was found that the largest efficient ring size is eight nodes and that raw network bandwidth seen by a processing element is limited at about 80Mbytes/s. A comparison to the DASH multiprocessor indicates that SCI's faster network yields better overall performance.

Keywords Scalable Coherent Interface, Rings, Multiprocessor, Performance evaluation, Simulation.

1 Introduction

In the quest for more affordable computing power, the Computer Architecture community has been paying a great deal of attention to multiprocessors built with logically-shared physically-distributed memory systems. Machines have been conceived, designed and built with sizes ranging from a handful to a few thousand (micro-)processors. One of the hurdles faced by designers of these machines is the network needed to interconnect the processors and memory since it must provide high-bandwidth low-latency path between them and must also support to the shared-memory model of programming.

The Scalable Coherent Interface (SCI) is an IEEE standard [12] that defines a network that, in theory, satisfies the above criteria. SCI defines a physical layer (cabling, clock frequencies), a logical communication protocol (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 (*e.g.* [18, 17]) and it does satisfy the high-bandwidth low-latency requirement.

This paper contains a performance evaluation of a complete multiprocessor based on an SCI interconnect where the influence of the cache coherence protocol on performance is investigated in detail. The experiments also relate the performance of the memory hierarchy to that of the interconnect. The influence of machine size, cache size and processor clock speed are assessed in turn. The topology studied is the ring and machine size ranges from one to 16 processors. The architecture simulator is driven with address sequences generated as a by-product of the execution of “real” programs. The workload consists of three programs from the SPLASH suite [19] (Cholesky, MP3D and Water) and three parallel loops (Gaussian elimination, matrix multiplication and all-to-all minimum cost paths).

The paper is organised as follows. Section 2 describes the simulation environment and the workload used to drive the architecture simulator. Section 3 contains a comparison between the DASH multiprocessor and an SCI machine with similar architectural parameters. Section 4 investigates the performance SCI rings.. Finally, Section 5 sums up the results of the simulations and presents conclusions. The Appendix gives a brief introduction to the communication and cache coherence protocols in SCI.

2 The Simulation Environment

The simulator consists of a memory reference stream generator and an 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 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 similar to those in [18, 17]. The model of the cache coherence protocol mimics the "typical set coherence protocol" as defined in [12]. For a description of the SCI communication protocol see [12, 17, 9]. For the description of the model of an SCI ring employed in the simulator, please see [10, 9].

The address sequences used to drive the simulator are generated by instrumenting parallel programs with Symbolic Parallel Abstract Execution (SPAE) [8]. 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 50 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 Workload. The workload used to investigate the behaviour of SCI multiprocessors consists of three parallel loops and three "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 Stanford's SPLASH suite [19]. 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. 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 minimising distortion caused by cold starts. Sizes were chosen so that there are at least 1.0×10^6 references to shared data. Table 1 shows the data-set sizes and the reference counts for the simulations.

`chol()` performs parallel Cholesky factorisation of a sparse matrix using supernodal elimination [19]. Cache size is one of the parameters used by the scheduler to allocate work to processors. The input matrix used is `bcsstk14` which contains 1806 equations and 30824 non-zeroes in the matrix and 110461 in the factor. The matrix `bcsstk14` occupies 420Kbytes unfactored and 1.4Mbytes factored.

`mp3d()` is a rarefied fluid flow simulator based on Monte Carlo methods [19]. The scheduling of tasks is static, synchronisation is based on barriers and granularity of work is large. Molecules are attached to processors rather than to spatial coordinates. The data set is scaled as $1.5 \times$ *nodes*. The simulation lasts 50 time steps.

`water()` is an n-body molecular dynamics program that evaluates forces and potentials in a system of water molecules in the liquid state [19]. The scheduling of tasks is static, synchronisation is based on barriers and granularity of work is large. The computation describing molecular motion involves a large number of array and floating point operations. The data set is scaled as $1.45 \times$ *nodes* . The system of molecules is simulated for 4 time steps.

`ge()` solves a system of linear equations by Gaussian elimination and backwards substitution. In this implementation, 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 consists of 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 $((n-k) \times (n-k+1))$. Input data set size grows as $1.26 \times$ *nodes*.

`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. This algorithm is also $O(n^3)$ and the input data sets are scaled up as $1.26 \times$ *nodes*.

`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 [5]. The vertices are labelled 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 outdegree 6. Input data set size is scaled as $1.26 \times$ *nodes*.

Machine size	1	2	4	8	16
Cholesky – `chol()`					
fixed size input: bcsstk14					
shared (% wr)	10.4 (18)	12.6 (23)	8.6 (23)	5.2 (23)	2.9 (19)
private (% wr)	31.0 (27)	8.5 (26)	2.7 (23)	1.0 (18)	0.9 (17)
instructions	71.7	37.0	20.3	11.6	8.1
MP3D – `mp3d()`					
molecules ($\times$1000)	3.0	4.5	6.7	10.1	15.2
shared (% wr)	5.4 (39)	5.5 (29)	4.5 (27)	5.0 (18)	6.0 (11)
private (% wr)	12.2 (18)	9.0 (18)	6.8 (18)	5.0 (18)	3.7 (18)
instructions	32.8	27.0	21.1	19.0	18.6
Water – `water()`					
molecules	54	78	113	163	237
shared (% wr)	1.4 (18)	1.5 (17)	2.2 (12)	2.9 (9)	2.9 (9)
private (% wr)	14.3 (19)	15.4 (19)	16.2 (19)	16.5 (19)	17.0 (19)
instructions	30.0	30.5	33.0	34.7	35.5
Gaussian elimination – `ge()`					
rows	136	171	216	272	343
shared (% wr)	2.6 (33)	2.6 (33)	2.6 (33)	2.5 (33)	2.5 (33)
private (% wr)	13.0 (7)	12.8 (7)	12.8 (7)	12.8 (7)	12.8 (7)
instructions	33.6	33.3	33.4	33.2	33.2
Matrix multiplication – `mmult()`					
rows	100	126	159	200	252
shared (% wr)	2.0(0.5)	2.0(0.4)	2.0(0.3)	2.0(0.2)	2.0(0.2)
private (% wr)	14.2 (14)	14.1 (14)	14.2 (14)	14.1 (14)	14.1 (14)
instructions	33.2	33.2	33.3	33.1	33.1
All-to-all minimum cost paths – `paths()`					
vertices	70	88	111	140	176
shared (% wr)	1.0(0.8)	1.0(0.6)	1.0(0.4)	1.0(0.3)	1.0(0.2)
private (% wr)	5.6 (6)	5.6 (6)	5.5 (6)	5.5 (6)	5.5 (6)
instructions	15.0	14.9	14.9	14.9	14.8

Table 1: Data-set sizes and per processor reference counts for the workload, in millions. 256Kbytes caches.

The proportion of references to shared data is only a small fraction of all memory accesses performed by the processor yet they sometimes account for a large fraction of the execution time. A program is said to be *processor bound* if the largest proportion of the execution time is spent performing instructions. Conversely, a program is *memory bound* when the largest fraction of the time is spent on data references.

3 Comparing DASH and SCI

The Directory Architecture for SHared memory (DASH) multiprocessor was conceived at Stanford University as a workbench for exploring the design of logically-shared physically-distributed memory multiprocessors [14]. A DASH prototype was built and its implementation and performance is discussed by Lenoski *et al.* in [15]. The availability of performance data makes possible a comparison between DASH and an SCI-based parallel machine with similar architectural parameters. However, because of intrinsic differences in architecture and run time environments, *i.e.* simulation compared to an actual machine, strict quantitative comparisons would be misleading. A qualitative comparison can nevertheless be informative. Table 2 shows the cost of individual memory references for the two architectures.

The DASH architecture. DASH consists of clusters of processing nodes interconnected by twin meshes. The clusters are bus-based multiprocessors within which cache consistency is maintained by a snooping protocol. Inter-cluster consistency is maintained by a full-directory invalidation protocol [14]. Each cluster contains four processors; each processor is connected to a 128Kbytes split primary cache (64Kbytes for instructions and 64Kbytes for data, write-through) and a 256Kbytes write-back secondary cache. Both caches are direct mapped and support 16-byte lines. The interface between primary and secondary caches consists of a 4-word deep write-buffer and a one-word read-buffer. Processor clock speed is 33MHz. The clusters also contain memory, the memory directory and inter-cluster communication interfaces. The interconnection network consists of two worm-hole routed networks, one for requests and one for responses. The latency through each node (network hop) is about 50ns. The peak bandwidth is 120Mbytes/s in and out of each cluster.

The SCI ring. The SCI ring was simulated with only one processor per node, with the same clock speed and cache hierarchy as DASH. The primary cache is split (64K + 64K), direct mapped, write-through. Secondary caches are 256Kbytes, direct mapped, write-back. Lines are 64 bytes wide. There is *no* write-buffer between the caches. Cache and memory latencies per word are the same as those in DASH – see Table 2. Notice that on the SCI ring, the latencies vary with ring size and, in the case of writes, the number of copies of the line. The processing nodes are interconnected by an SCI ring.

The workload for the comparison consists of `chol()`, `mp3d()` and `water()`. Data sets are, for `chol()` `bcsstk14`, for `mp3d()` 40000 molecules simulated over 5 time steps, and for `water()` 343 molecules simulated for two steps. In [15], Water is run with 512 molecules; the simulation time for that many molecules is so long as to be prohibitive. Thus, `water()` was simulated with a smaller data-set. The comparison should still be valid since, in the 16-node case, `water()` makes 2.6 million references per processor to shared data and that is enough to fill up the secondary caches at least a few times.

Cache operation	DASH	SCI ring
Read from primary cache	1	1
Fill from secondary cache	14	14
Fill from local memory	26	26
Fill from remote node	72	31–37
Fill from dirty-remote, remote home	90	49–60
Write owned by secondary cache	2	14
Write owned by local node	18	14
Write owned by remote node	64	63–207
Write to dirty remote, remote home	82	63–207

Table 2: Cache and memory operation latencies, in processor clock cycles.

Ring size	1	2	4	8	16
Cholesky – `chol()`					
read refs. (10^3)	8444	9883	6875	4638	2904
write refs. (10^3)	1862	2939	2075	1191	556
RD hit ratio	0.97	0.98	0.98	0.98	0.97
WR hit ratio	0.98	0.98	0.98	0.99	0.98
run time (s)	10.26	5.32	2.80	1.58	0.95
throughput (Mbytes/s)	0	1.18	2.06	2.54	2.69
MP3D – `mp3d()`					
read refs. (10^3)	3619	3755	1594	1041	1098
write refs. (10^3)	2343	1171	585	293	147
RD hit ratio	0.95	0.89	0.85	0.82	0.77
WR hit ratio	0.95	0.89	0.85	0.81	0.74
run time (s)	4.83	2.96	1.61	0.90	0.58
throughput (Mbytes/s)	0	9.5	15.8	19.9	20.6
Water – `water()`					
read refs. (10^3)	20172	12619	8163	4093	2414
write refs. (10^3)	3071	1536	768	384	192
RD hit ratio	0.96	0.96	0.88	0.84	0.83
WR hit ratio	0.99	0.99	0.95	0.93	0.92
run time (s)	41.96	21.20	10.90	5.50	2.81
throughput (Mbytes/s)	0	0.13	1.02	1.63	1.80

Table 3: Per node shared-data reference counts, secondary cache hit ratios, execution time and node throughput on the SCI ring.

Speedup. Figure 1 shows the speedup plots for both DASH and the SCI ring. Speedup data for DASH was taken from Figure 6 and Table 5 in [15]. The plot shows that `chol()` has a very similar speedup in both architectures. The differences for 2 and 4 nodes are related to better data mapping in the SCI ring. The differences on `mp3d()` are more pronounced. Network traffic is much higher than with the other two programs and, given SCI's higher bandwidth and lower latencies, it is not surprising that `mp3d()` scales

Figure 1: Speedup plots for `chol()` (left), `mp3d()` (mid) and `water()` (right).

Execution time breakdown. Figure 2 shows the execution time breakdown for the workload on the SCI ring. Time is split into: (1) busy time when the processor is performing instructions and operations on data; (2) the time wasted on read misses (RDmiss); (3) the time wasted on write misses (WRmiss); (4) the time spent on synchronisation actions (synchr). The plot also shows the total time spent on references to shared data (shared) and the time lost because of network latencies (network). The plots indicate that most of the stall cycles come from writes. This is partly because of the high latency of a write (14 cycles) and partly a "normal" feature of cache-coherent shared-memory multiprocessors [7]. The fraction of the time used up by `chol()` in references to shared data increases with ring size because of the relatively higher costs of remote references while the fraction due to instruction fetches becomes proportionally more important as the work per processor decreases. `mp3d()` spends 57–43% of the time on private references, a large fraction of which is on write misses. The fraction of time

Figure 2: Execution time breakdown for `chol()` (left), `mp3d()` (center) and `water()` (right). The lines show the fraction of shared data references and network latency.

4 SCI Ring Multiprocessors

The Simulated Multiprocessor. The multiprocessor consists of a number of processing nodes interconnected in a ring by SCI links. Each node contains a processor, a split primary cache, a coherent secondary cache, memory and an SCI interface. The CPU is a 32-bit SPARC processor that performs an instruction fetch and possibly a data read/write access on every clock cycle. The processor clock frequency is 100MHz. 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 size of the instruction cache (i-cache) and data cache (d-cache) is 8 Kbytes each (one page), both being 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 mapping of virtual to physical addresses is performed in parallel with primary cache tag-matching [16].

The secondary cache size is a simulation parameter. Sizes investigated are 64, 128, 256 and, 512Kbytes. Main memory is simulated as if implemented with DRAMs with 8-way interleaving. On all three levels of the memory hierarchy, cache and memory lines (blocks) are 64 bytes. The memory hierarchy satisfies the multilevel inclusion property [1] and the SCI coherency protocol actions affect only the secondary caches, thus called *coherent caches*.

The internal buses are 64 bits wide, except the processor-primary caches which are 32 bits wide. 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). Thus, a cache-to-memory read-line transaction costs 246ns for a 100MHz processor. To that, the network latency must be added if one end of the transaction, cache or memory, is at another node.

The sizes investigated are 64, 128, 256, 512 Kbytes and "infinite" caches. The size of caches should be chosen to minimise the miss ratios, that is, as large as possible, *and* to reduce the number of cycles the processor stalls waiting for memory references to be satisfied. The cache latency depends to a large extent on the memory technology and on the sophistication of the cache policies such as replacement, write-buffers, write-through/back. cache controller, the latency of the coherent caches was estimated to be three processor clock cycles. The access latency of DRAMs is of the order of tens of nanoseconds – 60 to 180 (in 1994), depending on size and organization of the memory array. When considering the overhead imposed by the coherency protocol, the latency of the memory was set to 120ns.

The Influence of Cache Size. Coherent cache size and tag access latency are two of the factors that have most impact on the performance of memory hierarchies. Figure 3 displays the execution time as a function of ring and cache size for the three SPLASH programs. Recall that the data-set sizes are scaled up to keep the work each processor does constant – see Table 1. For `chol()`, on a 4-node ring, the 128Kbytes cache is about 35% slower than the two larger sizes. The difference is not as pronounced for the other ring sizes. The 64Kbytes cache being faster than the 128Kbytes is due to an optimisation in `chol()`, by which the supernodes are chosen to fit the coherent caches. For all cache sizes (64-512Kbytes) and ring sizes 2–16, `mp3d()` has shared data hit ratios that are within one percentage point of one another. The same is true of the fraction of run time due to network latency, except that the interval is 4%. On a 16-node ring, `water()`'s shared data-set does not fit in the 64Kbytes caches. Hence the difference in execution time between the 64K and 128-512Kbytes coherent caches.

Figure 4 shows the relationship between cache and ring size and speed for the three parallel loops. Data-set sizes are scaled up with machine size – see Table 1. For `ge()`, the differences in run time are below 4% and this agrees with the rather small changes in shared data hit ratio with cache size. The performance of the system, when executing `mmult()`, improves with larger cache sizes. The improvement comes from a reduction in conflict misses and network delays. The 8-node machines endure higher instruction miss rates because of conflict misses. `paths()`, if the caches cannot accommodate the working set, the program speed is bound by the speed of the memory and ultimately by the network latency. For the 64Kbytes cache, the impact of the network latency increases dramatically with ring and data set sizes because of the poorer hit ratios.

Sharing-list length. In an SCI-based shared memory multiprocessor, data that is actively shared by processors is kept in linked lists, rooted at the data's home memory. When the data is to be updated, the list collapses, the data is updated and the sharing-list is eventually re-established. The collapsing of sharing-lists involves message exchanges between the processor at the head of list and each of the other nodes in the list. *Sharing-list length* is defined as the number of copies that have to be purged when a line is updated. The sharing-list length reflects the level of interference between processors on each other's computation. Because of the serialisation imposed by the coherence protocol, the cost of purging grows linearly with the length of the sharing-list. `paths()` has an average sharing-list length that grows roughly as $P/2$, for P processors. The other five programs have sharing-list lengths of one or less for ring sizes 2–8 and under 1.2 for 16-node rings. Sharing-list length is fairly independent of cache size. This is because most of the shared-data in `chol()`, `mp3d()` and `water()` is migratory in nature [21]. The same can be said of `ge()`, given its algorithm and simulation statistics.

Figure 3: Execution time as a function of cache size, for `chol()` (top), `mp3d()` (center) and `water()` (bottom). Time is broken down into network latency, references to shared-data and references to local data and instructions. Data sets grow with machine size.

Figure 4: Execution time as a function of cache size, for `ge()` (top), `mmult()` (center) and `paths()` (bottom). Time is broken down into network latency, references to shared-data and references to local data and instructions. Data sets grow with machine size.

Processor Clock Speed. Microprocessor technology is evolving at such a pace that the speed of processors, and indeed of workstations, doubles roughly every two or three years. What can be said about the performance of SCI, when the next generation of processors is introduced? Figure 5 shows the speedup attained by doubling the processor clock speed while keeping the other parameters unchanged. Note that coherent cache access latency is 3 processor clock cycles in both cases.

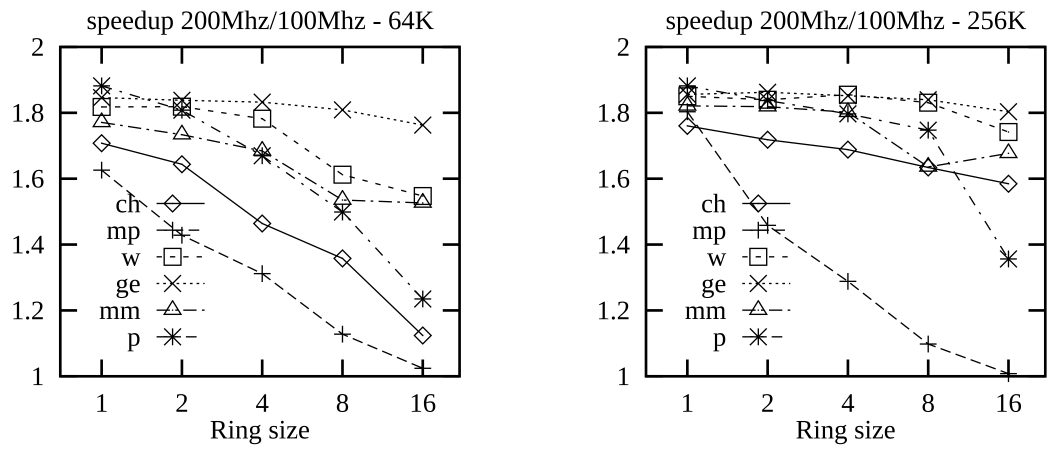

Figure 5: Speedup achieved by doubling processor clock frequency, with cache sizes of 64K (left) and 256Kbytes (right). 'ch' stands for `chol()`, 'mp' for `mp3d()`, 'w' for `water()`, 'ge' for `ge()`, 'mm' for `mmult()`, 'p' for `paths()`.

Some of the loss in speedup can be attributed to the relatively slower memory hierarchy, the influence of which can be gauged from the values for the uniprocessor – about 10 to 37% loss in speedup. Most of the loss in speedup for `chol()`, `mp3d()` and `paths()` is caused by network saturation. Plots of the ratio of link traffic for 100 and 200MHz processors are almost identical to those in Figure 5. Programs that generate low levels of network traffic can use a lot more bandwidth whereas programs that nearly saturate the ring suffer even higher round-trip delays with a faster rate of network requests.

Bandwidth and Latency. The transport mechanism of SCI is based on unidirectional point-to-point links. The simplest topology that can be implemented with these links 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 the *round-trip delay* of the ring. The number of packets a node can transmit per time unit depends on the traffic on the ring. The *traffic* seen by a node at its ring interface is defined as the number of symbols per time unit that is output by the ring interface. It consists of all the symbols passing through plus those inserted by the node itself. *Throughput* is the number of symbols per time unit inserted by the node and measures the amount of coherence-related traffic generated by the processor and cache/memory controllers.

	traffic				throughput				r-t delay			
size	2	4	8	16	2	4	8	16	2	4	8	16
`chol()`	8	30	72	144	6	10	12	12	37	46	72	130
`mp3d()`	65	198	383	621	46	64	62	50	37	48	83	173
`water()`	2	22	52	110	1	7	8	8	35	45	69	123
`ge()`	1	5	16	52	1	2	3	4	36	49	75	130
`mmult()`	1	6	20	118	1	2	4	10	33	52	74	132
`paths()`	8	24	70	395	5	9	12	34	36	50	78	156

Table 4: Link traffic (Mbyte/s), processor throughput (Mbyte/s) and round-trip delay (ns) for the workload.

Link traffic. Table 4 shows the traffic per link as a function of ring size. `mp3d()` and `paths()` produce high levels of traffic and suffer higher latencies. Traffic levels of 600 to 700Mbytes/s are a limiting factor in the performance of SCI-connected systems since, at these levels, round-trip delays are holding down the rate of network requests by processors. Bypass buffers endure utilisations of over 50% and that leaves few opportunities for injecting packets into the ring. The discussion about processor clock speed in Section 4 is clear evidence of the effects of network saturation: doubling processor clock rate does little to improve the performance of programs that are already driving the network into saturation.

Node throughput. Table 4 shows the effective throughput per node. Note that the measured throughput includes packet header overhead. Data throughput would be somewhat lower. The reason for including header overheads in the throughput measurement is that cache coherency commands are embedded in the packet headers and these comprise a large fraction of the information transferred by the cache coherence protocol.

Round trip delay. Table 4 shows the average round trip delay as a function of ring size. Note that latencies experienced accessing memory and caches are not included. The static latency for a 16-node ring is 116ns, for an average packet size of 11 symbols. `chol()`, `water()`, `ge()` and `mmult()` generate low network traffic and enjoy low latencies. `mp3d()` and `paths()` endure much higher latencies because of their higher throughput and increased network congestion.

Figure 6: Execution time breakdown for `chol()` (left), `mp3d()` (mid) and `water()` (right) – 100MHz clock.

Express Ring, KSR1 and Hector. In order to compute the cost of a remote transaction, memory and cache tag access latencies must be added to the round-trip delay. For the simulations reported here, the worst case is a cache-to-memory transaction: *ring latency*$+246ns$ ($30ns + 16ns$ plus $120ns + 80ns$). The best case is a cache-to-cache transaction, such as an invalidate transaction, costing *ring latency*$+60ns$ ($2 \times 30ns$). Barroso and Dubois, in [3], present simulation results for the Express Ring. The multiprocessor's interconnect is based on a slotted ring and cache coherence is maintained by a snooping protocol [2]. On a ring with 8 nodes, the shared-data miss latency for chol(), mp3d() and water() is between 280 and 320ns. On a 16-node ring, between 320 and 380ns and, on a 32-node ring, between 390 and 440ns. On 8-node rings, the shared-data miss latencies of an SCI ring are comparable to those of a slotted ring. On 16- and 32-node rings, the SCI ring would have higher latencies.

A comparison with the KSR1 [4] is difficult to make for lack of performance data on the applications employed here. It is likely the results would show the same broad tendencies as those of DASH since the two machines are built from similar technologies – SCI's faster network would provide a performance advantage. The Hector multiprocessor [20], using a hierarchical snooping protocol [6, 11] should have a performance comparable to that of the Express Ring. Holliday and Stumm report in [11], that Hector's hierarchical protocol scales well to a large number of processors (1024) if the applications possess good locality characteristics.

5 Conclusion

This paper contains a detailed performance evaluation study of SCI-based shared memory multiprocessors. Previous studies of SCI 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. The results of the simulations are summarised below.

A simple multiprocessor system was "implemented" in the simulator with components compatible with the current levels of performance. Several architectural parameters were investigated, namely machine size, secondary cache size and processor clock speed. Machines were simulated with one, two, four, eight and sixteen 100MIPS processors. In order to reproduce accurately the interleaving of the memory references in a NUMA architecture, the architecture simulator is driven by reference streams generated as a by-product of the execution of real programs. The simulated threads are scheduled for execution according to the state of the simulated multiprocessor and the actual delays incurred by references to remote memory and cache coherency actions.

Two of the programs in the workload are ill suited for execution on physically distributed memory. mp3d() has low hit ratios and its data is highly migratory, causing high levels of cache coherence activity and network traffic. This program exhibits poor performance in every published experiment seen by the authors. It is however very useful to expose architectural bottlenecks. The data used by `paths()` has a high degree of read-sharing and writes to shared data often cause the purging of long sharing-lists. This also causes high levels of network traffic and, for the smaller cache sizes, high levels of coherence activity. These two programs do drive the network into saturation and their performance is, in most of the experiments, limited by network bandwidth and delays. The other four programs have more regular behaviour and better coherent cache hit ratios. For them, the performance penalties imposed by the cache coherence protocol and interconnect are rather small. The overheads imposed by the cache coherence protocol are always smaller than 5% of the execution time. The losses caused by network latencies are under 10% of the execution time.

Node throughput is defined as the network bandwidth available to processing elements. For rings with processors and memory hierarchy as simulated, the experiments revealed that raw processor throughput is limited at about 80Mbytes/s because of network saturation. Data-only throughput is about 20 to 30% of raw throughput. Given that under 14% of all packets injected into the ring carry 64 bytes of data [9] while all except echo packets carry cache coherency information, raw throughput is a better measure of overall system performance.

High levels of network traffic cause queue backlogs in the link interfaces with round-trip delays increasing by as much as 25% as a consequence. For `mp3d()` and `paths()`, network saturation occurs for link traffic at 600 to 700Mbytes/s, for 8- and 16-node rings, and this in turn limits node throughput at 80Mbytes/s. The simulation results for SCI-rings indicate that, for hardware and software with characteristics similar to those investigated here, the maximum efficient ring size is between eight and sixteen. The scalability in these small systems (1, 2, 4, 8, and 16 processors) was found to be fairly good.

There is still work to be done in the performance evaluation of SCI based multiprocessors. Further investigation of small systems is needed since SCI is inexpensive enough to be used in small to medium sized machines. The evaluation of systems with hundreds of processors is necessary in order to assess the scalability of SCI-based multiprocessors. For simulations of large systems, the programs in the workload will need to be adapted, rewritten or replaced because they were designed and coded for medium size machines (32–64 processors). These codes are unlikely to scale up well to hundreds of processors without extensive rewriting. The simulation technique used here produces accurate results but its computational cost is very high. Two alternatives seem attractive. One is the direct simulation of the processors, thus avoiding interactions with the operating system. The other is to use a customisable synthetic workload. While not strictly realistic, proper tuning of parameters can produce insightful results.

Acknowledgements The source code for `ge()` was provided by Graham Riley, from the Centre for Novel Computing, Manchester University. The parallel version was compiled and optimised for the KSR1 at CNC. This work was partially supported by a scholarship from CAPES, Ministry of Education, Brazil.

References

[1] Jean-Loup Baer and Wen-Hann Wang. On the inclusion properties for multi-level cache hierarchies. In *Proc. 15th Int. Symp. on Computer Architecture*, pages 73–80, May 1988.

[2] Luiz A Barroso and Michel Dubois. Cache coherence on a slotted ring. In *Proc 1991 Int. Conf. Parallel Processing*, volume 1, pages 230–237, St. Charles, IL, USA, August 1991.

[3] Luiz A Barroso and Michel Dubois. The performance of cache-coherent ring-based multiprocessors. In *Proc. 20th Int. Symp. on Computer Architecture*, pages 268–277. ACM SIGARCH Comp Arch News 21(2), May 1993.

[4] H Burkhardt. Overview of the KSR1 computer system. Tech Report KSR-TR-9202001, Kendall Square Research, Boston, 1992.

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

[6] K Farkas, Z Vranesic, and M Stumm. Cache consistency in hierarchical ring-based multiprocessors. Tech Report EECG TR-92-09-01, Univ. of Toronto, 1992. Also in Proc. of Supercomputing '92.

[7] K Gharachorloo, A Gupta, and J Hennessy. Performance evaluation of memory consistency models for shared-memory multiprocessors. In *Fourth Int. Conf. on Architectural Support for Progr. Lang. and Oper. Sys.*, pages 245–257. ACM SIGARCH Comp Arch News 19(2), April 1991.

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

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

[10] Roberto A Hexsel and Nigel P Topham. The performance of SCI memory hierarchies. In *Proc of the Int. Workshop on Support for Large Scale Shared Memory Architectures*, pages 1–17, Cancún, Mexico, April 1994. In conjunction with 8th IPPS. Also Univ of Edinburgh Dept of Computer Science Tech Report CSR-30-94.

[11] Mark Holliday and Michael Stumm. Performance evaluation of hierarchical ring-based shared memory multiprocessors. *IEEE Trans. on Computers*, C-43(1):52–67, January 1994.

[12] IEEE. *ANSI/IEEE Std 1596-1992 – Standard for Scalable Coherent Interface.* IEEE, 1992. IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

[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] D Lenoski, J Laudon, K Gharachorloo, A Gupta, and J L Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proc. 17th Int. Symp. on Computer Architecture*, pages 148–159. ACM SIGARCH Comp Arch News 18(2), May 1990.

[15] D Lenoski, J Laudon, T Joe, D Nakahira, L Stevens, A Gupta, and J Hennessy. The DASH prototype: Implementation and performance. In *Proc. 19th Int. Symp. on Computer Architecture*, pages 92–103. ACM SIGARCH Comp Arch News 20(2), May 1992.

[16] O A Olukotun, T N Mudge, and R B Brown. Implementing a cache for a high-performance GaAs microprocessor. In *Proc. 18th Int. Symp. on Computer Architecture*, pages 138–147. ACM SIGARCH Comp Arch News 19(3), May 1991.

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

[18] Steven L Scott and James R Goodman. Performance of pipelined K-ary N-cube networks. Tech Report 1010, Computer Sciences Dept, Univ of Wisconsin–Madison, February 1991.

[19] J P Singh, W-D Weber, and 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).

[20] Z Vranesic, M Stumm, D Lewis, and R White. Hector: a hierarchically structured shared memory multiprocessor. *IEEE Computer*, 24(1):72–78, January 1991.

[21] Wolf-Dietrich Weber and Anoop Gupta. Analysis of cache invalidation patterns in multiprocessors. In *Third Int. Conf. on Architectural Support for Progr. Lang. and Oper. Sys.*, pages 243–256. ACM SIGARCH Comp Arch News 17(2), April 1989.

Appendix: The Scalable Coherent Interface

The description that follows concentrates on those features of SCI that are of relevance in this paper. SCI consists of three parts, the physical-level interfaces, the packet-based logical communication protocol, and the distributed cache coherence protocol. The physical interfaces are high speed unidirectional point-to-point links. One of the versions prescribes links 16 bits wide which can transfer data at peak speed of 1 Gbyte/s. 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 IO module or any combination of these. For most applications, a multiprocessor will consist of several rings, connected together by switches, *i.e.* nodes with more than one pair of link interfaces.

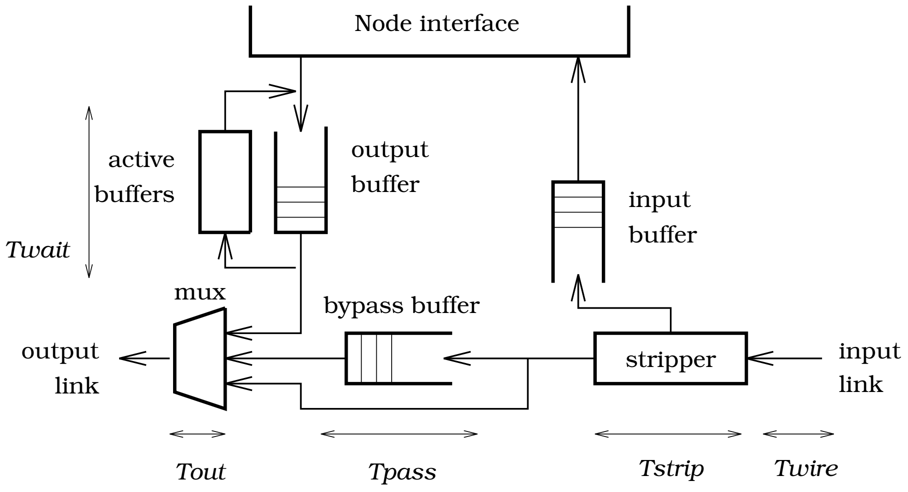

Figure 7: SCI link interface.

Logical Protocol. The *logical protocol* comprises the specification of the sizes and types of packets and of 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 is 4 symbols in length. A data packet carries 64 bytes of data.

The protocol supports two types of actions: requests and responses. A complete transaction, for instance, 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.

The network access mechanism used by SCI is the register insertion ring – see Figure 7. A node retains packets addressed to itself and forwards other packets to the downstream node. 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 at the bypass buffer and no packet is being forwarded from the stripper to the multiplexor. At the receiver node, the stripper parses the incoming packet and diverts it to the input buffer. On recognising a packet addressed to itself, the stripper generates an echo packet addressed to the sender and inserts it 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 packet.

It is likely that during the transmission of a packet, the bypass buffer will be filled with packets not addressed to the node. Once transmission stops, the node enters the *recovery phase* during which no packets can be inserted by the node. Each packet stripped creates spaces in the symbol stream. These spaces, called idle symbols, eventually allow the bypass buffer to drain, when new transmissions are possible. The protocol also ensures that the downstream nodes cannot insert new packets until the recovery phase is complete. This will cause a reduction in overall traffic and create enough idles to drain the bypass buffer.

When a packet is output, a copy of it is kept in an *active buffer*. If the status of its echo is ack, the original packet is dropped from the active buffer and the node can transmit another packet. If the echo carries a nack, the packet is retransmitted. This allows for one or more packets to be active simultaneously, *e.g.* one transaction initiated by the processor and other(s) initiated by the cache or memory controller(s). The number of active buffers depends on the type of the "pass transmission protocol" implemented. The options are: only one outstanding packet, one request-send and one response-send outstanding or, several outstanding request- or response-send packets.

Coherence Protocol. The SCI coherence protocol is based on a write-invalidate chained directory. Each cache line tag contains pointers to the next and previous nodes in the doubly-linked sharing list. A line's address consists of a 16-bit node-id and 48-bit offset. The storage overhead for the memory directory and the cache tags is a fixed percentage of the storage capacity. For a 64-byte cache block, the overheads are 4% at memory and 7% at the cache tags.

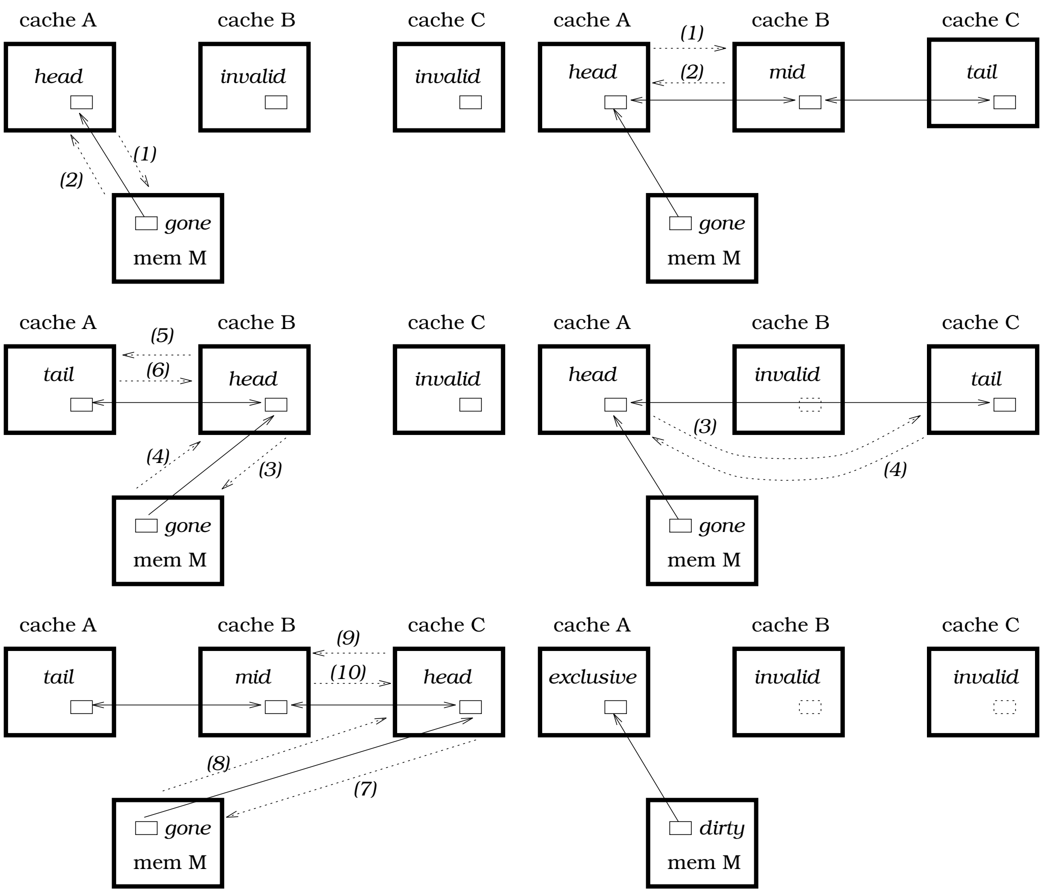

Figure 8: Sharing list setup (left) and purge sequence (right). Solid lines represent sharing list links, dotted lines represent messages.

Consider processors A, B and C, read-sharing a memory line L that resides at node M – see Figure 8. Initially, the state of the memory lines is home and the cache blocks are invalid. A read-cached transaction is directed from processor A to the memory controller M (1). The state of line L changes from home to gone and the requested line is returned (2). The requester's cache block state changes to the head state, i.e. head of the sharing list. When processor B requests a copy of line L (3), it receives a pointer to A from M (4). A cache-to-cache transaction, called prepend, is directed from B to A (5). On receiving the request, A sets its backward pointer to B and returns the requested line (6). Node C then requests a copy of L from M (7) and receives a pointer to node

B (8). Node C requests a copy from B (9). The state of the line at B changes from head to mid and B sends a copy of L to C (10). In SCI, rather than having several request transactions blocked at the memory controller, all requests are immediately prepended to the respective sharing lists. When a block has to be replaced, the processor detaches itself from the sharing list before flushing the line from the cache.

Before writing to a shared line, the processor at the head of the sharing-list must purge the other entries in the list to obtain exclusive ownership of the line – see Figure 8. Node A, in the head state, sends an invalidate command to node B (1). Node B invalidates its copy of L and returns its forward pointer (pointing to C) to A (2). Node A sends an invalidate command to C (3) which responds with a null pointer, indicating it is the tail node of the sharing list (4). The state of line L, at node A, changes to exclusive and the write completes. When a node other then the head needs to write to a shared line, that node has to interrogate the memory directory for the head of the list, acquire head status and then purge the other entries. If the writer is at the middle or tail, it first has to detach itself from the sharing list before attempting to become the new head.