The Performance of SCI Memory Hierarchies*

Roberto A Hexsel† & Nigel P Topham‡

Technical Report CSR-30-94
Department of Computer Science
Edinburgh University

February 1994

Abstract

This paper presents a simulation-based performance evaluation of a shared-memory multiprocessor using the Scalable Coherent Interface (IEEE 1596). The machines are assembled with one to 16 processors connected in a ring. The multiprocessor's memory hierarchy consists of split primary caches, coherent secondary caches and memory. For a workload of two parallel loops and three thread-based programs, secondary cache latency has the strongest impact on performance. For programs with high miss ratios, 16-node rings exhibit high network congestion whereas 4- and 8-node rings perform better. With these same programs, doubling the processor speed yields between 20 and 70% speed gains with higher gains on the smaller rings.

1 Introduction

The Scalable Coherent Interface (SCI) is an IEEE standard for high performance interconnects supporting a physically distributed logically shared memory [18]. SCI consists of physical interfaces, a logical communication protocol, and a distributed cache coherence protocol. The first silicon implementation of the protocols by Dolphin Technology, Norway, has been completed recently and some companies are already made public that SCI is part of forthcoming systems [15].

This paper presents the results of simulation experiments on a shared-memory multiprocessor based on SCI. The experiments investigate the two main components of SCI: the distributed cache coherence protocol and the packet based communication protocol. The impact of coherent cache size and latency, and processor clock speed on performance is assessed. The 90/10 locality rule states that "a program spends about 90% of its run time in 10% of its code"[16]. For a large number of parallel programs, the 10% are parallel loops that, for instance, solve a system of linear equations. Thus, the workload selected for the simulations consists of two programs based on parallel loops – Gaussian elimination and all-to-all minimum cost paths – and three thread based programs from Stanford's SPLASH suite [27], namely Cholesky, MP3D and Water.

The paper is organised as follows. Section 2 examines related work on both SCI and other shared-memory multiprocessors. Section 3 describes the simulator. Section 4 describes the workload and the simulation results are presented and discussed. Finally, conclusions are drawn in Section 5. The Appendix gives a brief introduction to SCI.

*To appear in Proceedings of the 8th International Workshop on Support for Large-Scale Shared Memory Architectures, Cancún, April 1994.

†rh@dcs.ed.ac.uk

‡npt@dcs.ed.ac.uk

2 Related Work

The quest for scalable cache coherent shared-memory multiprocessors has produced several cache coherence protocols and machine architectures [17]. To date, the KSR1 [9] is the only commercially available ring-based shared-memory multiprocessor. It is built as a hierarchy of rings and cache coherence is maintained by a snooping write-invalidate protocol. An important feature of the KSR1 is its memory hierarchy, composed only of primary and secondary caches, in what is called a Cache Only Memory Hierarchy (COMA). The KSR1 can scale up to 1088 processors in a two-level hierarchy of rings. The *ring:0* can accommodate 32 processors; the *ring:1* supports up to 34 *ring:0*'s. The remote access latency on a 32-node ring is under 7μs and, to reduce its effects, the KSR1 supports the software mechanisms prefetch and poststore.

Barroso and Dubois, in [5], present the design and simulation results for a slotted ring multiprocessor. They investigate two cache coherence protocols, one based on snooping and the other on a full-map directory. Their results indicate that the snooping protocol yields better performance. The maximum number of nodes that can be assembled on a slotted ring is limited to between 32 and 64. The directory based protocol yields miss latencies between 280 and 320ns on an 8-node ring, and between 310 and 380ns on an 16-node ring, for MP3D, Water and Cholesky [27].

Stanford's DASH is another example of a cache coherent shared memory multiprocessor [21, 22]. It consists of clusters of processors interconnected by a wormhole routed 2-D mesh. The memory coherence is maintained by a distributed invalidation directory-based protocol. The DASH, like SCI-based machines, is called a Cache Coherent Non-Uniform Memory Access Machine (CC-NUMA) because of the difference in access times for local and remote references.

The cache coherence protocol in SCI is a directory-based write-invalidate protocol. The directory implemented with doubly linked lists and allows for scaling up to 64K nodes. Communication is via unidirectional links and the basic topology is the ring. Higher dimensionality networks are implemented by having more than one SCI interface on each node. Scalability to 64K nodes comes at the price of added complexity in the communication and coherence protocols. For instance, a write to a shared datum needs a larger number of network messages for its completion than needed by the same operation in DASH [21]. Johnson, in [20], proposes additions to the cache coherence protocol to alleviate this problem. Additional links can be used in the linked lists, thus turning them into trees, and significantly improving the performance of invalidations when there is global sharing. Aboulenein *et.al*, in [1], examine SCI's hardware synchronisation primitive Queue On Lock Bit (QOLB). Its efficiency comes from it fitting in neatly with the linked-lists: waiting processes are naturally enqueued when they join the lock's sharing-list.

Data transport in SCI is based on pipelining data onto the network links. Scott and Goodman, in [25], investigate the performance of pipelined k-ary n-cube networks. In such a network, multiple bits may be traversing the same wire simultaneously. This makes the network's cycle time independent of wire length. When compared to synchronous networks (see [10, 2]), the pipelined networks yield lower latency and higher bandwidth, especially for high dimensional networks. The optimal dimensionality of pipelined networks is higher than that of synchronous networks and they should be grown by increasing the dimensionality while keeping the radix unchanged.

Scott *et.al*, in [24], and Scott in [26], present an analytical model of the SCI logical communication protocol. The model is based on M/G/1 queues and the ring is modeled as an open system. Their results indicate that the flow control mechanism is effective in preventing starvation and in reducing the effects of a hot transmitter on the ring. This mechanism is not as effective for non-uniform routing distributions. The maximum ring throughput is reduced by up to 30%, larger rings being more adversely affected. Read-request/read-response data-only ring throughput, for 64 byte data blocks, is around 800Mbytes/s (600Mbytes/s) on a 16 (4) node ring, fairly distributed among the nodes. They show that an SCI ring compares favourably to a bus.

3 The Simulator

The multiprocessor consists of processing elements (PE's) interconnected in a ring by SCI links. Each PE contains a processor, a split primary cache, a coherent secondary cache, memory and an SCI interface. The CPU is a 32-bit scalar Harvard processor that performs an instruction fetch and possibly a data read/write access on every clock cycle. The processor clock frequency is a simulation parameter and the values investigated are 100 and 200MHz. The size of the instruction cache (i-cache) and data cache (d-cache) is 8 Kbytes each, both being direct mapped. The data cache is write-through with no allocation of block 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 a simulation parameter. Sizes investigated are 64, 128, 256 and, 512 Kbytes. Memory is simulated as if implemented with DRAMs. On all three levels of the memory hierarchy, cache and memory lines (blocks) are 64 bytes wide.

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 10 ns 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 of the ends of the transaction, cache or memory, is at another node.

The memory model is sequential consistency [12]. The memory hierarchy satisfies the multilevel inclusion property [3]. So, the SCI coherency protocol actions affect only the secondary caches, thus called *coherent caches*. Coherency between primary and secondary caches is maintained by the cache controller. 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. Allocation of pages to nodes is naive: the first node that references a given page becomes its *home* memory. References to pages mapped to memory on other nodes are called *remote* references.

Simulation Methodology The 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 [25, 24, 23] but rather than using statistical analysis, traffic related values are measured and directly influence the behaviour of the simulated system. The model of the cache coherence protocol mimics the "typical set protocol" as defined in [18].

The address sequences used to drive the simulator are generated by instrumenting the programs (described in Section 4) with Symbolic Parallel Abstract Execution (SPAE) [13]. SPAE is based on the GNU gcc compiler and allows for tracing parallel programs at any desired level of detail. The resolution of the simulator is at instruction/data reference level. The cost of each memory reference is computed from the state of the system – level of network traffic and coherence actions performed – and those values are used to schedule the execution of the simulated processes/processors. Thus, the global interleaving of memory references is simulated with better accuracy than is possible with the method proposed in [23], at a higher computational cost however. Typically, a simulation run takes from 2 to 30 cpu hours on a lightly loaded Sparcstation2, depending on the data set size.

Model of the Ring Interface For the description that follows, please refer to Figure 1. The network clock cycle is 2ns (500MHz) and the physical links are 16 bits wide, in accordance with the SCI standard. The delay faced by a packet waiting to be transmitted ($Twait$) depends on the number and size of packets passing through the node. Likewise, the delay faced by packets at the bypass buffer ($Tpass$) depends on the frequency and size of packets inserted by the node. Wire propagation delay ($Twire$) is 2ns. The time to parse an incoming packet ($Tstrip$) and, the time to gate an outgoing symbol onto the output link ($Tout$) are also 2ns each. Thus, the delay

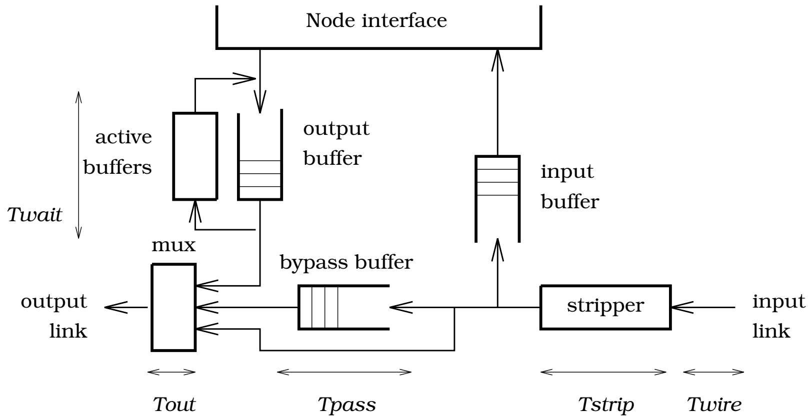

Figure 1: SCI ring interface.

involved in sending a packet from Node_A to Node_B and waiting for its echo can be computed by L_{AB}. To simplify the expressions, we omitted the modulus operations on summation indexes.

$$\begin{aligned} L_{AB,type} \;\; = \;\; & Twait_A + Tout \; + 2\, size(type) \\ & + \sum_{i=A+1}^{B-1} (\, Twire \; + Tstrip \, + Tpass_i + Tout \,) \\ & + Twire \; + Tstrip \; + Tpass_B + Tout \\ & + \sum_{i=B+1}^{A-1} (\, Twire \; + Tstrip \; + Tpass_i + Tout \,) \\ & + Twire \; + Tstrip \end{aligned}$$

Where *type* can be one of *Pcmd8*, *Pcmd16*, *Pdata*, *PdataX*, *Pecho* and, their sizes are 8, 16, 40, 48 and 4 symbols, respectively (1 symbol = 2 bytes). An *idle symbol* must precede each packet thus making the sizes 9, 17, 41, 49 and 5 in the throughput calculations. The term 2 *size(type)* is the time, in nanoseconds, needed to insert a packet into the ring. The peak bandwidth of a link or buffer is the maximum number of symbols that can pass through it per time unit. In the absence of traffic, peak bandwidth of the output or bypass buffer is 500 Msymbols/s (1Gbyte/s).

The average packet size through a link or buffer is $(Pavg \;\; = \sum_p f_p size(p) \,/\, \sum_p f_p)$ where $p \in \{Pcmd8,\ Pcmd16,\ Pdata,\ PdataX,\ Pecho\}$ and f_p is the frequency of packet type p. The throughput S of a buffer is the number of symbols that pass through it per unit of time: $(Sbuffer \;\; = \sum_p f_p size(p))$. The utilisation of a link or buffer is given by the throughput divided by the bandwidth available, times the average packet size. Thus, *Twait* is given by

$$Twait \; = Pavg_{\,tx}\, Stx \, / (BWmax - Spass)$$

and, *Tpass* is

$$Tpass \; = Pavg_{\,pass}\, Spass \, / (BWmax - Stx)$$

where *Spass* and *Stx* are the throughputs of the bypass and output buffers respectively, $(BWmax \; - Spass)$ is the bandwidth available at the output buffer, and $(BWmax \; - Stx)$ is the bandwidth available at the bypass buffer.

In the equation for the latency above, by making *Tpass* and *Twait* zero, the resulting equation yields the static latency of the ring, that is, it depends solely on propagation delays and is, in nanoseconds, $(6N + size(p))$ for N processors and packet p. Conversely, the dynamic component of the latency is obtained by considering only *Tpass* and *Twait*. The dynamic latency is estimated from the measured traffic. Buffer utilisation and average packet size are measured at $10\mu s$ intervals. Values from interval i are used to compute latencies during interval $i+1$.

The ring interface model assumes infinite input queues and does not account for the retransmission of packets dropped at their destinations. Since the memory is sequentially consistent, processors stall on remote references. However, cache or memory controllers may attempt to transmit response packets to complete outstanding transactions. The effect of more than one source of packets on a node is easily minimised by implementing at least two active buffers [24]. The model also ignores intranode contention, that is, the processor of a hot spot node does not see any contention for the internal buses and its local cache or memory.

The accuracy of this method lies between that of detailed simulation of the SCI communication protocol, where the simulator keeps track of every symbol travelling on the ring [8, 7, 24] and, that of trace postprocessing [23] or statistical analysis, where the network simulator is driven by random access patterns [6].

4 Simulation Results

The results of the experiments are presented in this section. First, the workload is presented and the behaviour of the programs discussed. Then, the following are examined in turn: influence of coherent cache size and latency, bandwidth and round-trip delay and, generation scalability.

Workload Input data was scaled up with ring size to keep the number of references to shared data per processor roughly constant. The simulations cause a minimum of 10^6 references to shared data. See Table 1 for the data-set sizes and Table 2 for the reference counts of each program. In all cases, tracing starts after initialisation.

Ring size		1	2	4	8	16
`chol()`			fixed size input			
`ge()`	(rows)	136	171	216	272	343
`mp3d()`	(molecules)	3000	4500	6750	10125	15187
`paths()`	(vertices)	70	88	111	140	176
`water()`	(molecules)	54	78	113	163	237

Table 1: Input data-set sizes.

Figure 2 shows the shared data hit ratio of all the programs for cache sizes of 64 and 256 Kbytes. Figure 3 shows the fraction of the execution time due to network latency, as computed by Equation 1. In those figures, ‘Ch’ stands for Cholesky, ‘Ge’ for Gaussian elimination, ‘Mp’ for MP3D, ‘P’ for all-to-all paths and ‘W’ for Water. 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.

Ring size	1	2	4	8	16
Cholesky – `chol()`					
shared refs 10^6 (% wr)	10.3 (18)	12.7 (23)	8.9 (22)	5.9 (20)	2.2 (14)
private refs 10^6 (% wr)	31.0 (27)	8.4 (26)	3.3 (21)	1.3 (14)	1.9 (21)
instructions 10^6	71.7	37.0	23.0	14.4	8.9
Gaussian elimination – `ge()`					
shared refs 10^6 (% wr)	2.6 (33)	2.6 (33)	2.6 (33)	2.5 (33)	2.5 (33)
private refs 10^6 (% wr)	12.9 (6.9)	12.8 (6.9)	12.8 (6.8)	12.7 (6.8)	12.7 (6.8)
instructions 10^6	33.7	33.2	33.4	33.2	33.2
MP3D – `mp3d()`					
shared refs 10^6 (% wr)	5.4 (39)	6.7 (24)	5.4 (22)	4.9 (18)	6.5 (10)
private refs 10^6 (% wr)	12.1 (17)	9.0 (18)	6.8 (18)	5.1 (18)	3.7 (18)
instructions 10^6	32.7	29.5	23.0	18.8	19.7
All-to-all minimum cost paths – `paths()`					
shared refs 10^6 (% wr)	1.0 (0.8)	1.0 (0.6)	1.0 (0.5)	1.0 (0.3)	1.0 (0.3)
private refs 10^6 (% wr)	5.6 (6.3)	5.5 (6.3)	5.5 (6.3)	5.5 (6.3)	5.5 (6.3)
instructions 10^6	15.1	14.9	14.9	14.9	14.7
Water – `water()`					
shared refs 10^6 (% wr)	1.4 (16)	1.6 (14)	2.0 (12)	2.8 (8)	5.5 (4)
private refs 10^6 (% wr)	14.3 (18)	14.2 (19)	15.2 (19)	15.3 (19)	15.1 (19)
instructions 10^6	28.4	28.5	30.1	32.6	37.6

Table 2: Per processor reference counts for the workload. 64K caches, 100 MHz.

`chol()` performs parallel Cholesky factorisation of a sparse matrix using supernodal elimination [27]. The scheduling of parallel work is done by a task queue and granularity of work is large. Cache size is one of the parameters used by the scheduler to allocate work to processors. The input data used is bcsstk14. For all ring sizes, `chol()` spends over 50% of the time executing instructions and, for ring sizes 2-8, over 20% of the time accessing shared data at the local cache and memory. For the 16-node ring, that falls to about 10%. Shared data hit ratios are always above 90%. So, for all ring sizes investigated, `chol()` is processor bound.

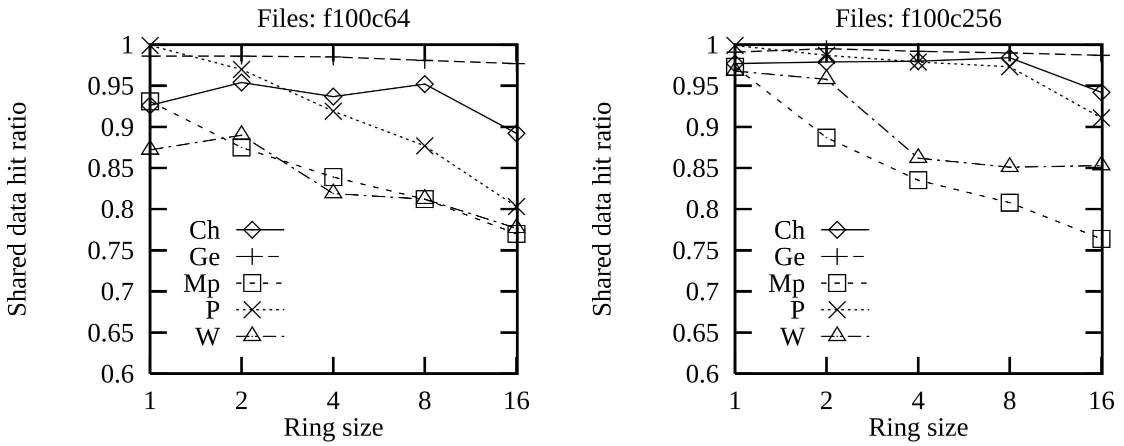

Figure 2: Shared data hit ratio, with cache sizes of 64 (left) and 256Kbytes (right).

`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 $(x = cx)$ followed by a rank-1 update of the matrix $(A = A + dxy)$ 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* . `ge()` spends over 67% of the time executing instructions, and 15% on shared data references. For all ring sizes (1-16), secondary cache hit ratios are above 97%. Thus, `ge()` is processor bound.

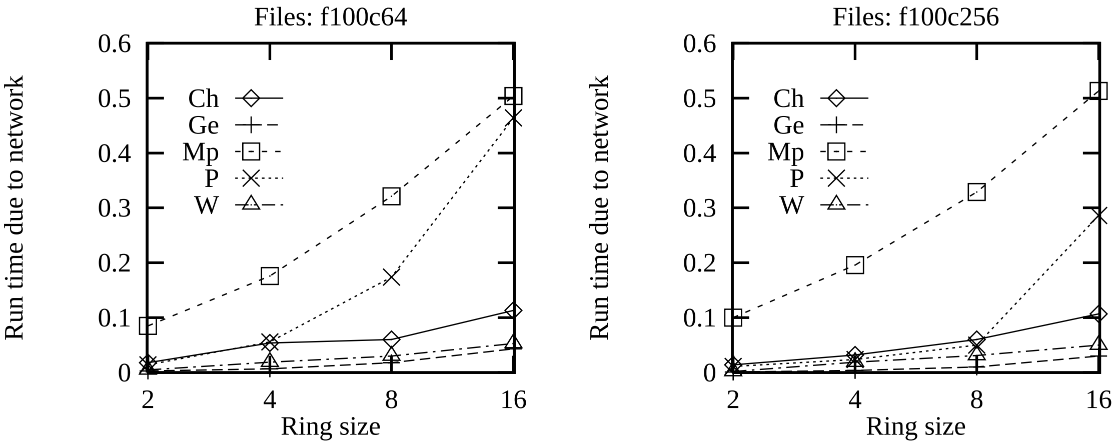

Figure 3: Fraction of execution time due to network latency, with cache sizes of 64 (left) and 256Kbytes (right).

`mp3d()` is a rarefied fluid flow simulator based on Monte Carlo methods [27]. The scheduling is static, synchronisation is based on barriers and granularity of work is large. The data set is scaled as $1.5 \times$ *nodes* . The simulation lasts 50 time steps. The uniprocessor spends 50% of the time at instructions, 30% on data that would be shared on a multiprocessor, and 23% of the time on private data. These figures, on a 16-node ring, fall to 10%, 5% and 5%, respectively – see Figure 5. The percentage of time spent on network latency climbs steadily from 0% to just over 50%. Thus, `mp3d()` is memory bound.

`paths()` is an instance 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 [11]. 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* . The code fragment below is the parallel loop where all of the work is done. `paths()`, on rings of up to 8 nodes, spends over 75% of execution time performing instructions, and about 10% on each of private and shared data references. For the 16-node ring and 256Kbytes cache, the shared data hit ratio is about 7 percentage points lower than on smaller rings and this in turn causes the time spend on communication latency to jump from under 5% to 28%. For a 64Kbytes cache, this last value is 47% – see Figure 2. For the data sets used here, `paths()` is weakly processor bound. If the shared data hit ratio falls to under 90%, it can easily become memory bound.

```
forall (t = 0; t < numProc; t++)
   for (k = start(t); k < end(t); k++)
      for (j = 0; j < rows; j++)
         for (i = 0; i < rows; i++)
            if (D[i,j] > (D[i,k] + D[k,j]))
               D[i,j] = D[i,k] + D[k,j];
```

`water()` is an n-body molecular dynamics program that evaluates forces and potentials in a system of water molecules in the liquid state [27]. The scheduling is static, synchronisation is based on barriers and granularity of work is large. The data set is scaled as $1.45 \times nodes$. The system of molecules is simulated for 4 time steps. `water()` spends over 50% of the time performing instructions and over 25% referencing private data. Even though shared data hit ratios aren't very high, less than 15% of the time is spent on shared data references. Thus, `water()` is processor bound.

Sharing-list length is defined as the number of copies that have to be purged when a line is updated. Because of the serialisation imposed by the coherence protocol, the cost of purging grows linearly with the length of the sharing-list. The sharing-list length reflects the level of interference between processors on each other's computation. `paths()` has an average sharing-list length that grows as $P/2$, for P processors. The other four 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.

Cache size and latency. Coherent cache size and latency can have a serious impact on performance. The effect of cache size is examined next. Figure 4 displays the execution time as a function of ring and cache size for `chol()` and `ge()`. For `chol()`, on a 4-node ring, the 128Kbytes cache is about 50% 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 `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.

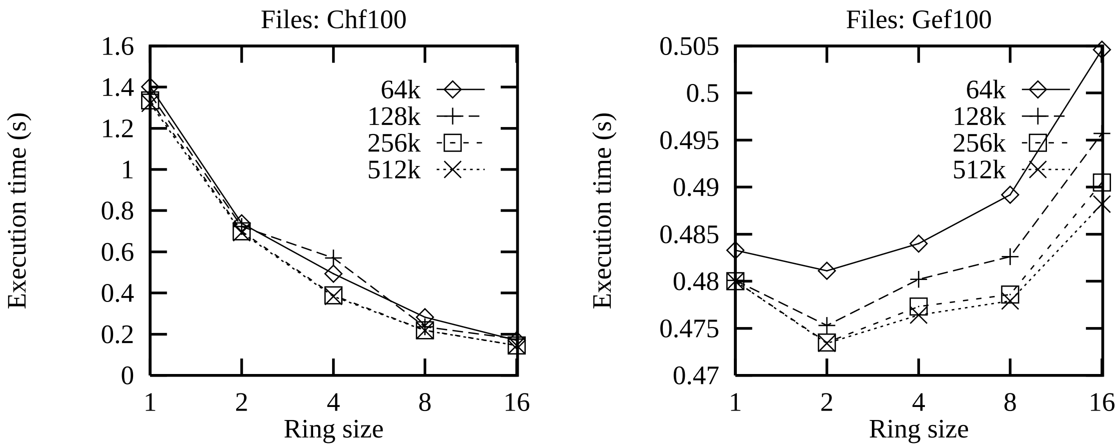

Figure 4: Execution time as a function of cache size, for `chol()` (left) and `ge()` (right).

Since `mp3d()` is memory bound, Figure 5 shows both the effects of cache size on speed, and the breakdown of the execution time by activity. For all cache sizes (64-512Kbytes) and ring sizes 2-16, the shared data hit ratios are within one percentage point. The same is true of the fraction of run time due to network latency, except that the interval is under 4%. The rings with smaller caches perform better because the distribution of page faults per node is less skewed than for the larger caches. If a more sophisticated mapping of pages to nodes were employed, the larger caches would be faster.

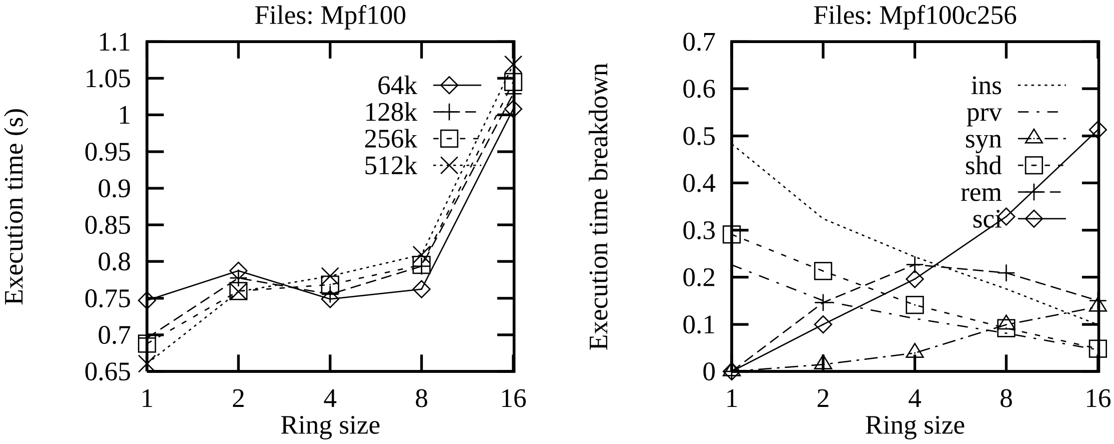

Figure 5: Execution time as a function of cache size, for `mp3d()` (left) and execution time breakdown (right). 'ins' stands for instructions, 'prv' for private data references, 'syn' for synchronisation (waiting at barriers and locks), 'shd' for local shared data references, 'rem' to cache and memory latencies on remote references and, 'sci' for network latency. The cost of remote references is ('rem'+'sci').

The influence of cache size on the performance of `paths()` and `water()` is shown in Figure 6. As discussed earlier, `paths()` is a borderline program: 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 – see Fig. 3. `water()` is less dependent on cache size. On a 16-node ring, the difference in execution time between the 64Kbytes and 256/512Kbytes is caused by an increase in the miss ratio.

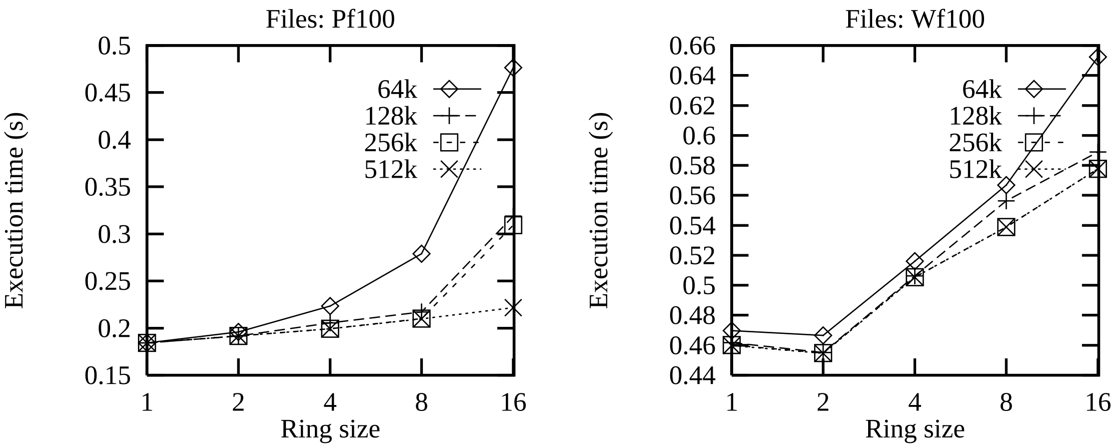

Figure 6: Execution time as a function of cache size for `paths()` (left), and `water()` (right).

Table 3 shows the effects on execution time of changing one of the major design parameters while keeping the other two constant. The basis for comparison is a system with 256Kbytes coherent caches with 3 processor cycles of access latency and memory access latency of 120ns. The factor that has the most influence is the cache access latency (between -12% and $+14\%$) while memory access latency has the least influence (between -6% and $+6\%$). The small effect of increasing cache size to 512Kbytes is related to the already high hit ratios attained with the 256Kbytes caches. For the workload studied here, caches with 2 processor cycles of access latency yield an average 8.5% speed improvement while, on average, the speed loss can be 9.1% (8.6%) on the 4-node (8-node) ring with a 4-cycle latency coherent cache. The system designers have to weight the cost increase against the speed gains. The plots in Figures 4 and 6 provide evidence against the use of 64Kbytes secondary caches. The more conservative cache latency of 3 cycles was adhered to for the experiments.

Nodes	4						8					
change:	c size		c latency		m latency		c size		c latency		m latency	
	128	512	2 cy	4 cy	80	160	128	512	2 cy	4 cy	80	160
`chol()`	1.15	1.01	0.88	1.14	0.98	1.00	1.06	0.99	0.87	1.13	0.97	1.04
`ge()`	1.01	1.00	0.93	1.07	1.00	1.00	1.01	1.00	0.93	1.07	1.00	1.00
`mp3d()`	1.02	0.98	0.92	1.08	0.94	1.06	1.00	1.02	0.95	1.08	0.96	1.05
`paths()`	1.03	1.00	0.94	1.07	0.99	1.01	1.04	1.00	0.94	1.06	1.00	1.01
`water()`	1.00	1.00	0.91	1.10	0.99	1.01	1.03	1.00	0.90	1.09	0.99	1.01
average	1.04	1.00	0.92	1.09	0.98	1.02	1.03	1.00	0.92	1.09	0.98	1.02

Table 3: Sensitivity of execution time to variations in cache size, cache latency and memory latency. The basis is 256Kbytes, 3 processor cycles and 120ns, respectively.

Bugge *et.al*, in [8], compare the performance of three uniprocessor memory architectures, two of which based on a 32- and on a 64-bits wide Futurebus$^+$. The third employs SCI links between secondary cache and memory. Their trace-driven simulation results indicate that with a time-shared multiprogramming workload, secondary cache *size* has the largest impact on the performance of the memory hierarchy whereas the influence of tag access latency is small.

Throughput and round-trip delay. The behaviour of SCI's transport mechanism is investigated next. Figure 7 shows the throughput per node, that is, the number of bytes per time unit inserted in the output buffer by the processor and cache/memory controller. The average packet size varies from 36.0 to 43.4 bytes, smaller rings carrying larger packets. Also, smaller caches generate more of the smaller packets that carry the coherency commands.

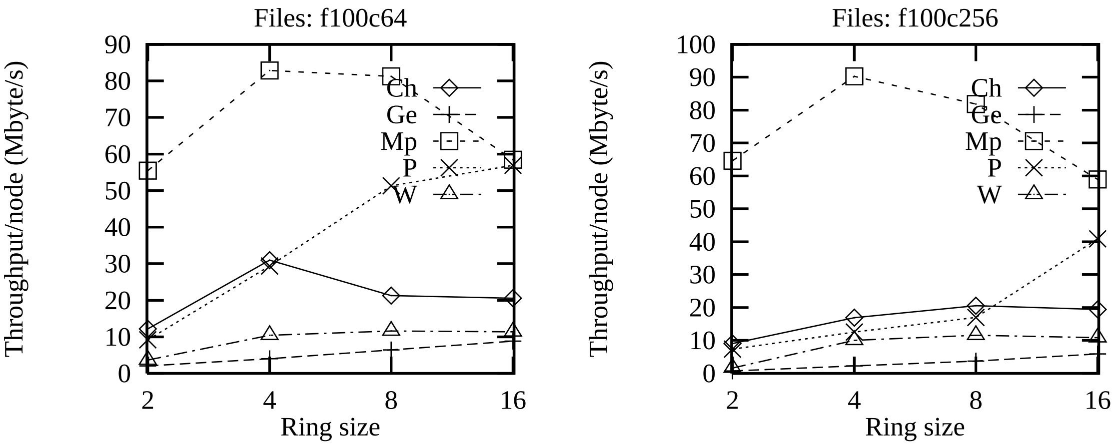

Figure 7: Throughput per node, with cache sizes of 64 (left) and 256Kbytes (right).

Figure 8 displays the effective throughput as a fraction of the throughput offered. If the bandwidth were infinite, the effective throughput would be limited only by static delays on the network whereas on the actual interconnect it is limited by network congestion. `mp3d()` on a 16-node ring and 256Kbytes cache has a shared-data hit ratio of 77% which makes its throughput high. It would be even higher were the network not in saturation.

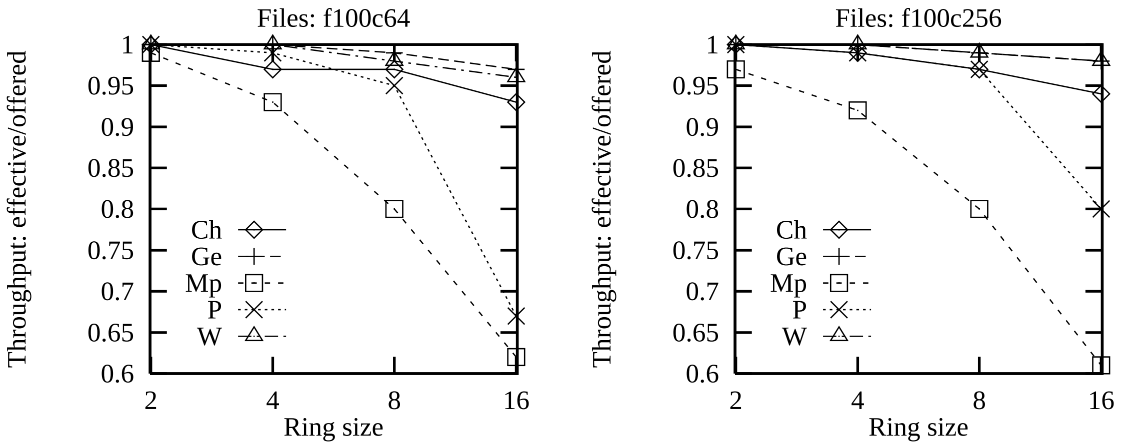

Figure 8: Per node effective throughput as a fraction of throughput offered, with cache sizes of 64 (left) and 256Kbytes (right).

Figure 9 shows the average round trip delay as a function of ring size. This delay is the time elapsed from inserting a packet in the output queue until its echo is stripped by the sender. Note that latencies experienced accessing memory and caches are not included. The static latency for a 16-node ring is 136ns, for an average packet size of 20 symbols. `chol()`, `ge()` and `water()` generate low network traffic and enjoy low latencies. `mp3d()` and `paths()` endure much higher latencies because of their higher throughputs and increased network congestion.

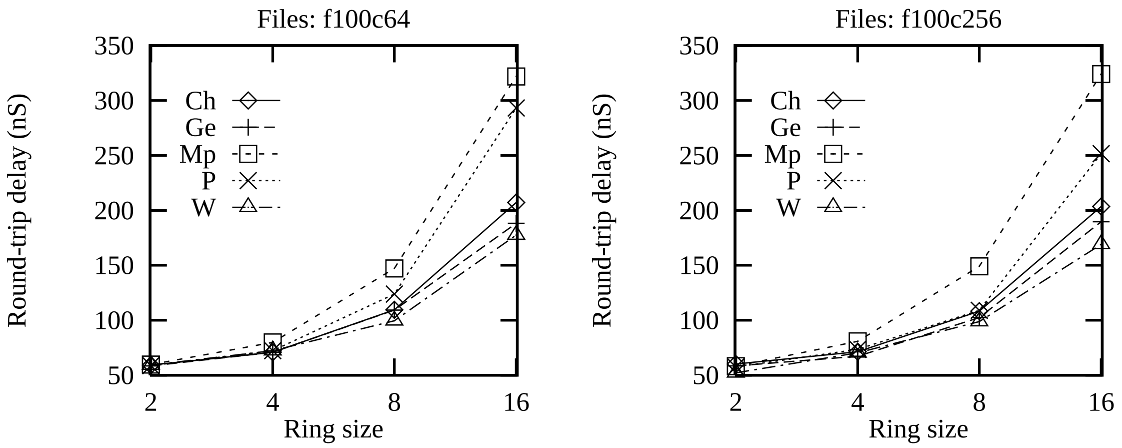

Figure 9: Average round-trip latency, with cache sizes of 64 (left) and 256Kbytes (right).

The throughput and round-trip delay for `mp3d()` are in agreement with those predicted in [24]. There are some differences in the underlying models and assumptions though. Here, the simulator uses 5 packet sizes rather than 3 and the proportion of 80 byte packets (`mp3d()`, 16 nodes) is 25% rather than 40%. The machines simulated here do not behave like an open system since the processor stalls on remote references. However, cache and memory controllers can and do transmit while the processor is idle.

In order to compute the cost of a remote transaction, memory and cache tag access latencies must be added to the round trip delay. 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 [4], present simulation results of a full directory coherence scheme based on a slotted ring. 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.

Figure 10 shows the traffic per link as a function of ring size. The traffic consists of the packets inserted by a node and the packets passing through that node addressed to downstream nodes. `mp3d()` and `paths()` produce high levels of traffic and suffer higher latencies. The plots in Figures 7, 9, and 10 provide evidence that SCI rings do not scale well past 8 nodes for programs that have poor locality or high levels of data sharing.

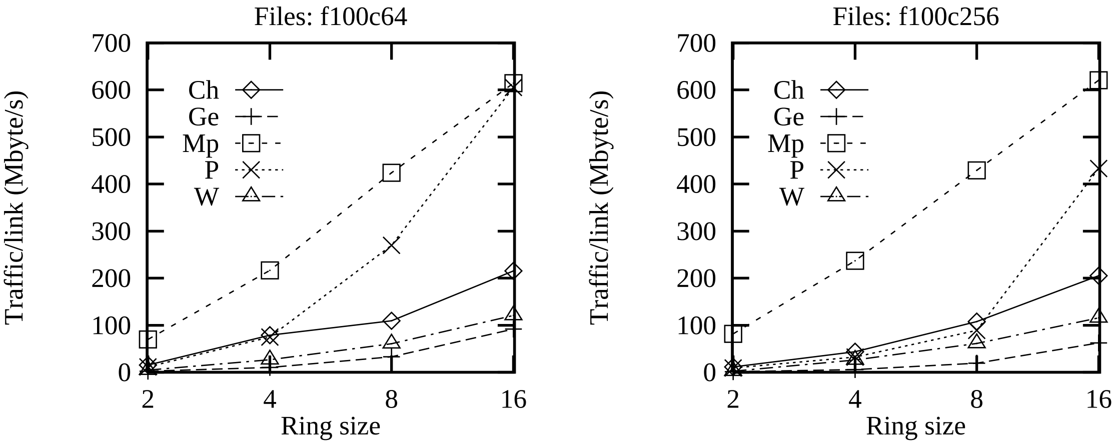

Figure 10: Traffic per link, with cache sizes of 64K (left) and 256Kbytes (right).

Bogaerts *et.al*, in [7], present simulation results for 10-node rings and a multi-ring system with 1083 nodes for data acquisition applications in particle physics. They concentrate on the bandwidth consumed by SCI *moveXX* transactions for DMA and ignore coherence related events. For DMA *move256* transactions (move 256 bytes with no acknowledgement), the bandwidth is about 175 Mbytes/s per node. When fair bandwidth allocation is employed, that figure is 125 Mbytes/s. Also, in the context of bulk real-time data acquisition and preprocessing, the largest adequate ring size is 10 nodes.

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 11 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.

Some of the loss in speedup can be attributed to the relatively slower memory and intranode buses and its influence of this can be gauged from the values for the uniprocessor. As discussed earlier, for a 100MHz clock, an increase of 30% in memory latency slows execution down by up to 6%, `chol()` and `mp3d()` being the worst affected. Most of the loss in speedup for `chol()`, `mp3d()` and `paths()` is caused by saturation of the network. Plots of the ratio of link traffic for 100 and 200MHz processors are almost identical to those in Figure 11. Programs that use little bandwidth can use a lot more whereas programs that nearly saturate the ring suffer even higher round-trip delays with a faster clock.

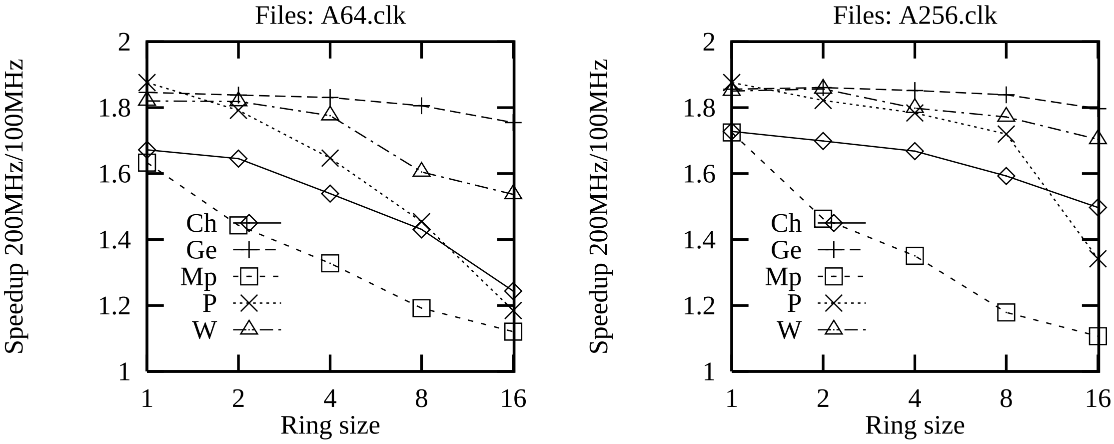

Figure 11: Speedup achieved by doubling processor clock frequency, with cache sizes of 64 (left) and 256Kbytes (right).

5 Conclusion

This paper presents the results of detailed simulation of multiprocessors based on SCI rings. The simulator was driven with address traces produced on-the-fly from five scientific applications. These consist of two parallel loops, Gaussian elimination and all-to-all minimum cost paths, and three thread based programs from the SPLASH suite: Cholesky, MP3D and Water.

The influence of secondary cache size and latency and of memory latency were investigated. For the workload chosen, rings with 2 to 16 processors and cache sizes of 64, 128, 256 and 512Kbytes, it was found that secondary cache latency has a stronger impact on performance than cache size. 64Kbytes caches proved to be too small for the data set sizes employed. Memory latency has the smallest, but non-negligible, impact on execution time.

Of the five programs, only MP3D and all-to-all paths needed high bandwidths, 'paths' only achieving high throughputs with 64Kbytes of cache and its lower hit ratios. The throughputs achieved by MP3D on 4- and 8-node rings were between 80 and 90Mbytes/s. On 16-node rings, its throughput fell to about 55Mbytes/s because of the high network traffic (over 600Mbytes/s per link). Its round-trip delays were about 50% higher than the other programs. Rings with 16 nodes seem to saturate at the load levels caused by MP3D. All programs except Water show poorer performance on a 16-node ring when compared to 4- or 8-node rings. This indicates that, for the workload used, the maximum efficient ring size is 8 nodes.

The clock frequency of microprocessors roughly doubles every two or three years. The use of faster processors increases the throughputs of the programs and their demands on the network. The experiments shown that, with a processor clock and secondary cache twice as fast, rings that are saturated with slow processors will be even more saturated with faster processors. Programs that are not near to saturating the network achieve high speedups.

The continuation of the work described here comprises of the simulation of higher dimensional networks. These will consist of rings of 4 or 8 nodes interconnected by switches. It is anticipated that the high computational costs will limit the scope of investigation somewhat.

Acknowledgements The authors would like to thank Stuart Anderson and Todd Heywood for helpful comments on an early version of this paper. The referees provided valuable comments on contents and presentation. Graham Riley, at CNC, Manchester University, supplied the parallel version of Gaussian elimination. Roberto Hexsel is partially supported by a grant from CAPES, Ministry of Education, Brazil.

References

[1] N M Aboulenein, S Gjessing, J R Goodman, and P J Woest. Hardware support for synchronization in the Scalable Coherent Interface (SCI). Tech Report 984, Univ of Wisconsin–Madison, February 1990.

[2] Anant Agarwal. Limits on interconnection network performance. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

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

[4] Luiz A Barroso and Michel Dubois. The performance of cache-coherent ring-based multiprocessors. Technical Report CENG-92-19, Dept of Electrical Engineering–Systems, Univ of Southern California, LA, November 1992. To appear in ISCA'93.

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

[6] L N Bhuyan, D Ghosal, and Q Yang. Approximate analysis of single and multiple ring networks. *IEEE Trans. on Computers*, C-38(7):1027–1040, July 1987.

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

[8] H O Bugge, E H Kristiansen, and B O Bakka. Trace driven simulations for a two-level cache design in open bus systems. In *Proc. 17th Int. Symp. on Comp. Arch.*, pages 250–259. ACM SIGARCH Comp Arch News 18(2), May 1990.

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

[10] William J Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. on Computers*, 39(6):775–785, June 1990.

[11] Narsingh 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.

[12] Kourosh Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proc. 17th Int. Symp. on Comp. Arch.*, pages 15–26. ACM SIGARCH Comp Arch News 18(2), May 1990.

[13] 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.

[14] David B Gustavson. The Scalable Coherent Interface and related standards projects. *IEEE Micro*, pages 10–22, February 1992.

[15] David B Gustavson. First SCI chips coming. Electronic mail message posted to `sci-announce@hplsci.hpl.hp.com`, January 1993.

[16] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1990. isbn 1-55860-069-8.

[17] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, 1993. isbn 0-07-031622-8.

[18] IEEE. *Scalable Coherent Interface, P1596/D2.00.* IEEE, November 1992.

[19] David V James et al. Scalable Coherent Interface. *IEEE Computer*, 23(6):74–77, June 1990.

[20] Ross Evan Johnson. Extending the Scalable Coherent Interface for large-scale shared-memory multiprocessors. Tech Report ????, Univ of Wisconsin–Madison, 1993. PhD thesis.

[21] Daniel Lenoski et al. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proc. 17th Int. Symp. on Comp. Arch.*, pages 148–159. ACM SIGARCH Comp Arch News 18(2), May 1990.

[22] Daniel Lenoski et al. The DASH prototype: Implementation and performance. In *Proc. 19th Int. Symp. on Comp. Arch.*, pages 92–103. ACM SIGARCH Comp Arch News 20(2), May 1992.

[23] Daniel Menascé and Luiz A Barroso. A methodology for performance evaluation of parallel applications on multiprocessors. *Journal of Parallel and Distributed Computing*, 14(1):1–14, January 1992.

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

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

[26] Steven Lee Scott. Toward the design of large-scale, shared-memory multiprocessors. Tech Report 1100, Univ of Wisconsin–Madison, 1992. PhD thesis.

[27] 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).

Appendix: The Scalable Coherent Interface

The description that follows concentrates on those features of SCI that are of relevance in this paper. For more details, please see [18, 19, 14]. 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. 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 switches, *i.e.* nodes with more than one pair of link interfaces.

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. Figure 1 shows a block diagram of the ring interface. A node retains packets addressed to itself and forwards the 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, 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 then 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 – for details see [18, 24].

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 a write-invalidate chained directory scheme. 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 address offset. The storage overhead for the memory directory and the cache tags is a fixed percentage of the total storage capacity. For a 64-byte cache block, the overhead at memory is 4% and at the cache tags 7%.

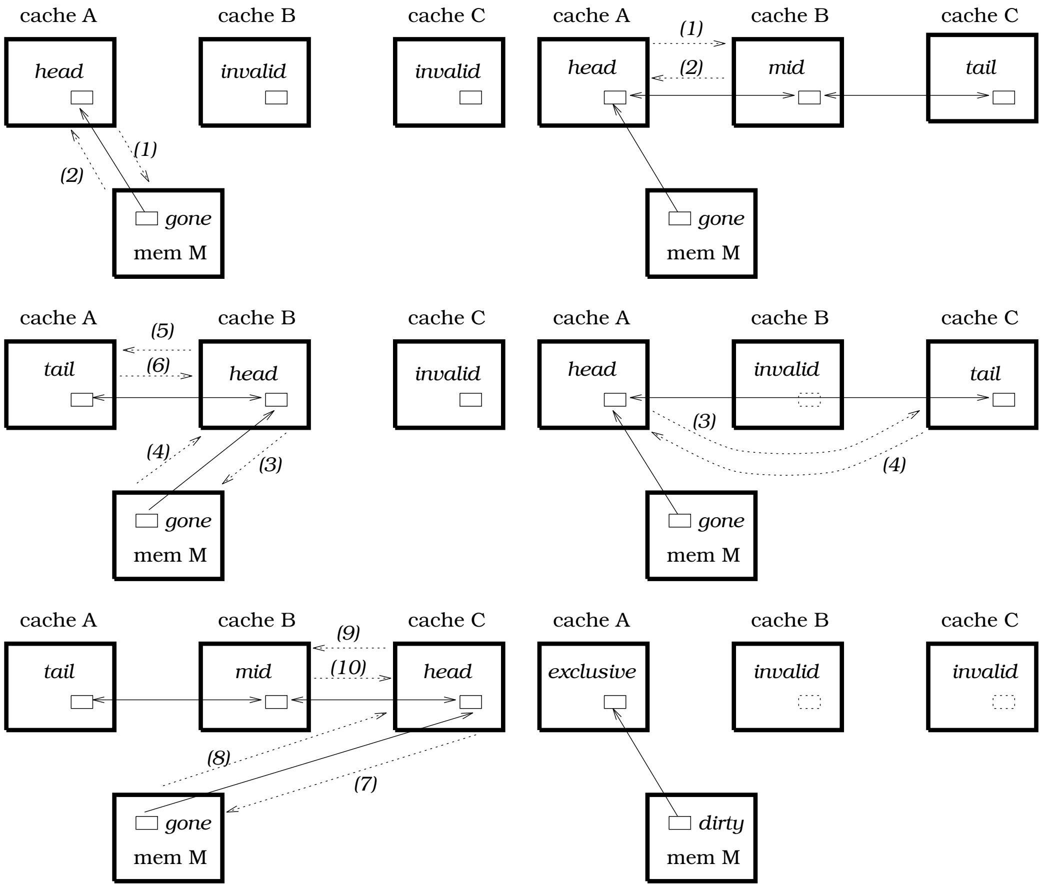

Figure 12: Sharing list setup (left) and purge sequence (right). Solid lines represent sharing list links, dotted lines represent messages.

Consider processors A, B and C, read-sharing a memory line L that resides at node M – see

Figure 12. 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 12. 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.