A Stochastic Model of Cache Coherency Overhead in SCI Rings

A.J. Field and P.G. Harrison
Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ U.K.

Abstract

We present a new analytical performance model of the IEEE P1596 Standard Coherent Interface, which is a distributed cache coherency protocol for shared memory multiprocessors. We focus upon an implementation of the protocol on a unidirectional ring architecture (the "default" architecture for SCI systems). We identify the possible memory and cache line states and corresponding processor actions for a memory access and derive the equilibrium line state probabilities by solving a Markov model expressed as a set of fixed point equations. The probabilities of a processor performing a particular action then follow, from which the message transmission profile for each processor is derived. These traffic equations are then fed into an M/G/1 model for the ring architecture in which the ring traffic at a node has priority over traffic originating in that node. Further analysis then leads to the mean message transmission time, and hence the mean memory access time, and processor utilisation. We illustrate the application of the model by undertaking a performance comparison of two alternative node architectures and report some numerical results for various parameterisations.

1 Introduction

In this paper we present an analytical model of a ring-based shared-memory multiprocessor operating the IEEE P1596 Standard Coherent Interface (SCI) [6, 5, 15, 11]. We demonstrate the use of this model by comparing the performance of two internal node architectures when the protocol is being used as an interconnect medium for multiple bus-based multiprocessors.

We seek to develop an analytical model of SCI for efficient, and low-cost, experimental analysis of different design options. Various simulation models have been developed for SCI, the most detailed of which is the full chip-level specification (in ANSI C) published as part of the IEEE standard. However running this as a simulation is intractably slow as it essentially emulates the behaviour of each SCI chip on each clock cycle. The execution time of simulation programs using this code runs into weeks for any realistic applications and as such it is not viable as an experimental design aid. More abstract and efficient stochastic simulations have been produced, for example that of [15]; this is computationally less expensive than the C code specification, but produces only approximate performance measurements and still at considerable expense in time. More recently an execution-driven simulation has been developed [16] which can run real applications, including the SPLASH suite codes [18], under a simulated memory system. This execution-driven model yields accurate performance figures but it does not explicitly model the ring. The execution times are also very long since every memory reference is compiled to a call to the simulated memory. In contrast the model we present can assimilate performance graphs for a range of parameter settings in a few minutes using the Mathematica tool [12].

To date, the only analytical model relevant to SCI that we are aware of is that of [14] which uses an M/G/1 model for analysing slotted ring networks, suitably adapted for a subset of the

SCI traffic. The paper does not address the important issue of SCI cache coherency, however, and so does not model the message traffic generated by each node to maintain a coherent global memory. In this paper we develop a detailed model of the SCI caches and memories from which we can determine the all-important coherency traffic, which we later incorporated into an M/G/1 model of the ring. To do this we appeal to the techniques descibed in [7, 8] for modelling bus-based coherency protocols; this is based upon an anlysis of the cache line states and associated state transitions. Our model is tailored to the line states of SCI and differs further from [7] in its characterisation of memory usage. Our model of the memory/cache access patterns has been derived based on preliminary experimental analysis of parallel program behaviour on a conventional shared-memory computer [9]. This represents an improvement on existing models of cache systems which have predominantly assumed a uniform memory reference model—in practice, however, the distribution of shared memory references is distinctly non-uniform, largely due to synchronisation traffic which is both intensive and highly localised in some applications.

The final model we present is currently being used as a design aid to enable engineers to predict the performance of a wide range of SCI implementation options. With the basic SCI model in place, the effects of different parameterisations of the specific hardware and software architecture described here, and of different hardware implementations of the protocol, can be analysed with relative ease.

The rest of the paper is organised as follows: Section 2 describes the core principles underlying SCI and defines a simplified version of SCI which we seek to model. This incorporates all the salient features of SCI whilst avoiding low-level details which might render an analytical model intractable. Section 3 describes out new model for SCI. This is essentially derived by combining two modelling approaches, namely that of [7] for single-bus systems and [14] for a simple SCI ring. Section 4 presents some sample numerical results from the model and discusses model validation. Finally, Section 5 presents a summary of the paper and some proposed future work.

2 Overview of SCI

The Scalable Coherent Interface, or SCI, is a high performance interface standard for distributed shared-memory multiprocessors [5, 6, 11]. The default communication network is a unidirectional ring, which we model here, although other topologies are permitted by the standard, see [2].

Data is transferred undirectionally around the ring in packets which are either *send* packets or *echo* packets.

A send packet is placed into the transmit queue and is sent out on the output link when there is nothing to transmit from either the input link or ring buffer. The send packet is *passed* through all intermediate nodes towards its destination. At each intermediate node the link data has priority over data in the transmit queue.

The sharers of each cache line are maintained by a doubly-linked *sharing list*; the principal is similar to that of directory-based protocols [1] except that the directory is distributed over the processors' memories and caches. Directory-based systems are typified by DSMM architectures such as the Stanford DASH multiprocessor [13, 4] and as first proposed by [3].

Each node contains a cache and a portion of the shared memory which is divided into *blocks* of 64 bytes. The caches are addressed in units of one block (called cache *lines*). Cache lines carry state information and two pointers to maintain the sharing list for the line.

Figure 1 shows a typical node configuration (we consider an alternative configuration in our analysis later on). Non-shared (i.e. private) data held locally is *not* duplicated into the SCI cache when accessed locally since there can be at most one copy of such data. Non-local private data [1] is cached locally when requested but the cache entry does not contain sharing list information for the same reason.

New sharers are added to an existing sharing list following a read miss and existing sharers are deleted as the results of a displacement. The protocol is *invalidation-based* which means that a

[1] *Data private to a process may become detached from the process as a result of task migration, for example.*

sharer must invalidate all other copies of a line before it can write to it—this requires the sharing list to be *reduced* to a singleton list before the write can proceed.

3 A New Model for SCI

Although the SCI cache coherency protocol is directory-based and not snoopy-based, the method used in [7] can still be applied since each cache line undergoes similar transitions, although the states are somewhat different, referring to the doubly-linked list of shared lines in different nodes.

The approach we have taken in the model described below combines the two approaches by using the method in [7] to obtain the equilibrium line state probabilities and using these to determine the message arrival rates for an M/G/1 ring model as used in [14].

We will assume that a memory reference from a processor is a read with probability α and a write with probability $1-\alpha$. Each read/write request emanating from the local processor(s) is sent first to the local cache and we define β to be the probability of a cache hit so that $1-\beta$ is the probability of a miss.

In order to be able to vary the sharing behaviour in the workload model, we need to be able to characterise an application's use of memory. We will assume that a memory access is either to *private* or *shareable* memory and that shareable memory is further classified into *control variables* and *shared areas*. Control variables comprise synchronisation variables like semaphores and locks for controlling access to shared structures; these variables are typically addressed with high frequency. The shared areas contain larger data structures, typically with access priveliges governed by control variables, that are relatively infrequently accessed. As the two types of shareable memory behave very differently it is important to distinguish them in the model. We will denote by γ_1 and γ_2 the proportion of memory accesses that are to control structures and shared areas respectively, and γ_3 the proportion of private memory accesses, so that $\gamma_1 + \gamma_2 + \gamma_3 = 1$.

Another factor concerns the use of private data. Typically, private data addressing is biased toward the local memory of a node. That is, private data is more likely to reside in the local node memory than it is to reside anywhere else. For this reason it is necessary to introduce a skew parameter, σ, which is the probability that a private memory access is to a location held in the local memory. In this situation, no ring traffic is produced. Under a completely uniform addressing mechanism this probability would ordinarily be the reciprocal of the number of nodes.

To complete the parameterisation we define: K—the number of nodes, N—the total number of shareable memory blocks, m—the number of (shareable) lines containing control variables, and n—the capacity (in blocks) of each local cache.

An analysis of memory access behaviour in shared-memory systems suggests that m is small in comparison to n and that the address mapping is such that all m control variables can reside in a cache at any time [9]. This means that the cache now has two distinct regions: one of size m blocks which can hold (local copies of remote) private data and both types of shared data, and one of size $n-m$ blocks which can hold only private and shared region data. We will consider each region separately in calculating the equilibrium cache line state probabilities as these will be different for the two regions.

A processor in an SCI system alternates *think periods* and periods when it waits for a memory access to be serviced. Let the think period have a mean of $1/\tau_0$. After a think period, the processor generates a memory request and this request may or may not invoke a transaction across the ring. During the time a request is processed, the processor is idle. We aim to determine the probability, π, that a processor is busy doing useful work. Note that a memory request can be satisfied locally either by the local memory itself (for private data) or the SCI cache (for cached non-local private data or any locally cached shared data), or a combination of the two (for uncached shared data held in the local memory which must be cached having been read from). The local access times will thus vary for each type. Memory requests which cannot be satisfied locally will involve communication with other nodes in accordance with the SCI protocol.

Cache Line States

A memory block may be in two possible states

1 Home The block is not cached by any processor

2 Cached The block is cached by one or more processors

The caches hold local copies of memory blocks and these copies may either be *clean* or *dirty*. A clean copy is the same as the home copy of the block. A dirty copy differs from the home copy and becomes so following a write operation. To enable us to model realistic program behaviour we consider *private* and *shareable* data separately. Private data is used by only one processor and is typically located in the local memory of that processor. Shareable data is assumed to be evenly distributed over the global address space and can be read and written by any processor.

We can thus identify eleven different line states although eight of them we will subdivide further in accordance with the type of data they contain. This is explained below:

1 Private Clean The location contains a clean copy of private (non-shareable) data

2 Private Dirty The location contains a dirty copy of private (non-shareable) data

3 Only Clean The location contains a clean copy of a memory block and is alone in the sharing-list.

4 Only Dirty The location contains a dirty copy of a memory block and is alone in the sharing list.

5 Head Clean As 3 but the location is at the head of a list containing at least two members.

6 Head Dirty As 5 but the cached copy is dirty. All members of the list hold the same dirty copy, but this is different to the home memory block.

7 Mid Clean The location is in the middle of the list (i.e. at neither the head nor tail); the cached copy is clean.

8 Mid Dirty The location is in the middle of the list (i.e. at neither the head nor tail); the cached copy is dirty.

9 Tail Clean As 5 but the location is at the tail of the sharing list.

10 Tail Dirty As 6 but the location is at the tail of the sharing list.

11 Invalid The location contains no usable information.

Cache partitioning

The workload model assumes that certain cache lines may contain shareable control variables, in addition to shared region data and non-local private data, whilst others may contain only the last two. We will number these regions I and II respectively and will consider them separately in the model, making the important observation that the entries of a sharing list must all be in the same region of their corresponding caches. This property applies to all n-way set associative caches. Within each region, the cache lines are now statistically identical.

In order to produce the correct transition rates for Region I we must further tag the states to distinguish the two type of shareable data; we thus annotate states 3–10 with the subscript 'a' if the (shared) block contains control information and 'b' if it contains shared region data. We make the assumption that different types of data do not co-exist on the same cache line; this is easily ensured by a suitable allocation of global memory addresses. Thus, for example, state 5_a indicates a clean line forming the head of a sharing list which contains control variables, e.g. locks or semaphores. Similarly state 8_b indicates a dirty line in the middle of a sharing list which contains variables from the shared region. The cache lines in Region I therefore have nineteen possible states, and those of Region II eleven.

3.1 Actions generated by a processor

In order to develop an analytical model of the system we need to identify the various processor actions affecting the state of a sharing list. In what follows the distinguishing subscripts 'a' and 'b' are omitted as the actions are the same for each region:

Creation (CR) A read or write miss on a location in state 11 not cached by other processors. The processor sends 1 short message to memory and receives 1 long answer from memory containing a copy of the block. The location transits to state 1, 2, 3 or 4 depending on whether the request was to a private or shareable block and whether it was a read or write.

Addition (AD) A read miss on a location in state 11 but cached by other processors. The processor sends 1 short message to memory, which sends one short message to the old list head; it receives one long message from the old head containing the block. The location transits to state 3 or 4 depending on whether the existing sharing list is clean or dirty.

Deletion-Creation (DC) A read or write miss on a *home* block mapping to a cache location previously in state 3–10. The processor deletes the old entry from the sharing list, therefore sending 1 or 2 short messages (depending on its location in, and length of, the sharing list), and then creates a new sharing list. As before it sends 1 short and receives 1 long. As before, the location transits to state 1, 2, 3 or 4.

Deletion-Addition (DA) A read miss on an already *cached* block mapping to a location in state 3–10. The processor deletes the old entry from the sharing list (involving 1 or 2 short messages as above) and adds itself to the head of the new one (2 short messages and 1 long one), the location transits to state 3 or 4.

Deletion-Reduction (DR) A write miss on a *cached* block mapping to a location in state 3–10. The processor deletes its entry from the sharing list, therefore sending 1 or 2 short messages, takes the head of the needed list involving 2 short message and 1 long message, and then sends (and receives) short messages – one for each sharing list entry – to reduce the list to a single sharing list.

Take-Reduction (TR) A write hit on a block mapping to a location in state 7–10. The processor deletes itself from the sharing list (involving 1 or 2 short messages as above), then takes the head of the same list sending 2 short messages and receiving one. It then reduces the list to a singleton as above. This case is distinguished from the above because no long messages need be sent: the current copy of the block is already cached.

Head-Reduction (HR) A write hit on a location in the processor's cache but in state 3,4. The processor sends (and receives) short messages to reduce it to a singleton sharing-list as above.

Invalid-Reduction (IR) A write miss on a *cached* block mapping to a location in state 11. The processor adds itself to the sharing list and reduces it. The location transits to state 4.

Note that for each Deletion operation, if the location was in state 2 then a *writeback* of the line will also be performed, requiring one additional long message. The writeback is required to update the master copy of the line at the home memory.

3.2 The analytical model

We assume that the evolution of the state of a given location in a cache follows a Markov process, independent of the states of other locations. This process is irreducible, aperiodic and has a finite state space and thus has a steady-state. A separate Markov process is established for each cache region for the reasons already stated.

Let q_j be the steady state probability that a given cache location in Region I is in state j for $j = 1, 2, 3_a, 3_b, ..., 10_b, 11$. It is also the average proportion of cache locations in state j in equilibrium. Let q'_j be the same for Region II, $j = 1, ..., 11$.

According to the possible actions generated by a processor, we derive the instantaneous transition rates from one state to another. A transition may be created by an action of either the local processor or a remote processor. The steady-state probabilities for each region will then be obtained by solving the resulting system of balance equations.

State Transitions

The state transitions occur as a result of read and write operations on the caches. We imagine we are looking at a particular line of a cache and consider the various read/write operations which can change the state of that line; we shall refer to this as the *observed* line. A *local* read/write is one issued by the processor local to the node containing the observed cache line and a *remote* read/write is one issued by one of the other processors in the machine. For remote operations we distinguish reads/writes to the same global address as is cached in the observed line from those to addresses which coincidentally map to the same cache location. The latter can lead to sharing list deletions, specifically if there is a miss on a remote cache line containing another copy of the observed line. We assume here a direct-mapped cache so that all entries in a given sharing list are held at the same address within their respective caches; multiple associativity is easily modelled, however.

Note that a "sharing list" here is defined to contain only shareable blocks. For private blocks there can be at most one cached copy of the home block and we shall refer to these simply as "private copies".

In order to write down the transition rates we define the following probabilities for Region I:

- p_{1_r} is the probability for a location in state 7/8 (Mid) to be just after the head of the sharing list
- p_{2_r} is the probability for a location in state 7/8 to be just before the tail
- p_{3_r} is the probability for a head to be followed by just a tail

where r is either a or b. We similarly define p'_1, p'_2 and p'_3 for Region II. These can not be determined exactly but we can approximate them by making assumptions about the lengths of sharing lists. In what follows, we consider Region II as the equations are shorter. The corresponding equations for Region I are derived similarly.

Now consider a non-exclusive sharing list and let L' and $L'_{mid} = L' - 2$ be the random variables denoting the total number of elements in the list, and the length of the 'mid-zone' of the list respectively. In order to calculate the above pobabilities we make the assumption that L'_{mid} has a truncated geometric distribution with parameter $0 \leq u' \leq 1$ so that

$$P'_l = \Pr\{L'_{mid} = l\} = \left(\frac{1-u'}{1-u'^{K-1}}\right) u'^l$$

¿From the steady-state probabilities we can determine the mean length of a sharing list to be $D' = (q'_5 + q'_6 + q'_7 + q'_8 + q'_9 + q'_{10})/(q'_5 + q'_6)$ (in fact since $q'_5 + q'_6$ is not exactly the same as $q'_9 + q'_{10}$ because of the approximations in the model we take an average of the two in performing this calculation). Given D' we can then determine the required value of u'. Firstly,

$$D' \quad = \quad 2 + \sum_{l=0}^{K-2} \frac{(1-u')lu'^l}{1-u'^{K-1}}$$

Manipulating this equation yields:

$$u' = \frac{D' - 2 + (K - D' + 1)u'^{K-1} \; - \; (K - D')u'^K}{D' - 1}$$

which is solved iteratively. This now enables us to estimate the probabilities listed above:

$$\begin{aligned} p'_1 = p'_2 &= \sum_{l=1}^{K-2} \frac{1}{l} \Pr\{L'_{mid} = l \mid L'_{mid} > 0\} \\ &= \frac{1-u'}{u'-u'^{K-1}} \sum_{l=1}^{K-2} \frac{u'^l}{l} \\ p'_3 &= \Pr\{L'_{mid} = 0\} \\ &= \frac{1-u'}{1-u'^{K-1}} \end{aligned}$$

Similar variables P_l, u_r and D_r where $r = a, b$ are defined for Region I.

Note that we could store the sharing list length explicitly in the state of shared lines in order to model more accurately the state transitions which take place in practice. However, this causes a dramatic increase in the number of states (K-fold asymptotically) and still requires assumptions to be made about the distribution of the sharing list length whenever the observed line is added to another list.

Transition Rates

We are now is a position to write down the transition rates. In every case we have divided by the (default) factor $\pi\tau_0$ which is the rate at which memory requests are submitted by a processor. Note that the transition rates do not cover transitions from a state to itself, for example as a result of a read hit in states 1–10. We begin with Region I and introduce the following notation:

$$\begin{aligned} s &\equiv \text{state variable} \\ a &= \{3a, ..., 10a\} \\ b &= \{3b, ..., 10b\} \\ \overline{a} &= \{1, 2, 11\} \cup b \\ \eta_a &= 1 - (1 - \sum_{i \in a} q_{i_a} + q_{i_b})^{K-1} \\ \eta_b &= \frac{q_{3_b} + q_{4_b} + q_{5_b} + q_{6_b})\ n\ (K-1)}{(N-m)} \\ \epsilon_r &= \frac{q_{3_r} + q_{5_r}}{q_{3_r} + q_{4_r} + q_{5_r} + q_{6_r}} \end{aligned}$$

where $r = a, b$. The quantity $\eta_r, (r = a, b)$ is the probability that, given a cache miss on a type r memory block, that block is cached somewhere else in the machine. ϵ_r is the probability that a cached copy of a type r line is clean. η_b is given by the total number of cached shared-region blocks divided by the total number of shareable memory blocks and $1 - \eta_a$ is the probability that a given control variable line is not cached by any other node.

The transition rates are therefore as follows (excluding transitions from any state to itself):

$$\begin{aligned} \{a, b, 11\} \rightarrow 1 &\qquad \frac{\alpha\gamma_3}{n} \\ 2 \rightarrow 1 &\qquad \frac{\alpha(1-\beta)\gamma_3}{n} \\ s \rightarrow 2 &\qquad \frac{(1-\alpha)\gamma_3}{n} \\ \overline{a} \rightarrow 3a &\qquad \frac{\alpha\gamma_1(1-\eta_a)}{m} \\ \{5a, 9a\} \rightarrow 3a &\qquad \frac{p_{3_a}(1-\gamma_1)}{n} \end{aligned}$$

$$
\begin{aligned}
\{1,2,11,a\} &\rightarrow 3b && \frac{\alpha\gamma_2(1-\eta_b)}{n} \\
b &\rightarrow 3b, b \neq 5b, 9b && \frac{\alpha(1-\beta)\gamma_2(1-\eta_b)}{n} \\
\{5b,9b\} &\rightarrow 3b && \frac{\alpha(1-\beta)\gamma_2(1-\eta_b)}{n} + \frac{p_{3_b}(1-\beta)}{n} \\
s &\rightarrow 4a, s \neq 6a, 10a && \frac{(1-\alpha)\gamma_1}{m} \\
\{6a,10a\} &\rightarrow 4a && \frac{(1-\alpha)\gamma_1}{m} + \frac{p_{3_a}(1-\gamma_1)}{n} \\
s &\rightarrow 4b, s \neq 6b, 10b && \frac{(1-\alpha)\gamma_2}{n} \\
\{6b,10b\} &\rightarrow 4b && \frac{(1-\alpha)\gamma_2}{n} + \frac{p_{3_b}(1-\beta)}{n} \\
\overline{a} &\rightarrow 5a && \frac{\alpha\gamma_1\eta_a\epsilon_a}{m} \\
7a &\rightarrow 5a && \frac{p_{1_a}(1-\gamma_1)}{n} \\
\{1,2,11,a\} &\rightarrow 5b && \frac{\alpha\gamma_2\eta_b\epsilon_b}{n} \\
s &\rightarrow 5b, s \in b, s \neq 7b && \frac{\alpha(1-\beta)\gamma_2\eta_b\epsilon_b}{n} \\
7b &\rightarrow 5b && \frac{\alpha(1-\beta)\gamma_2\eta_b\epsilon_b}{n} + \frac{p_{1_b}(1-\beta)}{n} \\
\overline{a} &\rightarrow 6a && \frac{\alpha\gamma_1\eta_a(1-\epsilon_a)}{m} \\
8a &\rightarrow 6a && \frac{p_{1_a}(1-\gamma_1)}{n} \\
\{1,2,11,a\} &\rightarrow 6b && \frac{\alpha\gamma_2\eta_b(1-\epsilon_b)}{n} \\
s &\rightarrow 6b, s \in b, s \neq 8b && \frac{\alpha(1-\beta)\gamma_2\eta_b(1-\epsilon_b)}{n} \\
8b &\rightarrow 6b && \frac{\alpha(1-\beta)\gamma_2\eta_b(1-\epsilon_b)}{n} + \frac{p_{1_b}(1-\beta)}{n} \\
5a &\rightarrow 7a, 6a \rightarrow 8a && \frac{(K-D_a)\alpha\gamma_1}{m} \\
5b &\rightarrow 7b, 6b \rightarrow 8b && \frac{(K-D_b)\alpha\gamma_2}{N-m} \\
3a &\rightarrow 9a, 4a \rightarrow 10a && \frac{(K-1)\alpha\gamma_1}{m} \\
3b &\rightarrow 9b, 4b \rightarrow 10b && \frac{(K-1)\alpha\gamma_2}{N-m} \\
7a &\rightarrow 9a, 8a \rightarrow 10a && \frac{p_{2_a}(1-\gamma_1)}{n} \\
7b &\rightarrow 9b, 8b \rightarrow 10b && \frac{p_{2_b}(1-\beta)}{n} \\
a &\rightarrow 11 && \frac{(K-1)(1-\alpha)\gamma_1}{m} \\
b &\rightarrow 11 && \frac{(K-1)(1-\alpha)\gamma_2}{N-m}
\end{aligned}
$$

For example, $8b \rightarrow 6b$ describes a transition from a dirty shared region line in the mid state to the same in the head state. This occurs either on a local read miss on a dirty shared region variable

	Actions							
State	CR	AD	DC	DA	DR	TR	HR	IR
1,2	0	0	$\overline{\eta}\overline{\beta}$	$\alpha\eta\overline{\beta}$	$\overline{\alpha}\eta$	0	0	0
3,4	0	0	$\overline{\eta}\overline{\beta}$	$\alpha\eta\overline{\beta}$	$\overline{\alpha}\eta$	0	0	0
5,6	0	0	$\overline{\eta}\overline{\beta}$	$\alpha\eta\overline{\beta}$	$\overline{\alpha}\eta$	0	$\overline{\alpha}\beta$	0
7,8	0	0	$\overline{\eta}\overline{\beta}$	$\alpha\eta\overline{\beta}$	$\overline{\alpha}\eta$	$\overline{\alpha}\beta$	0	0
9,10	0	0	$\overline{\eta}\overline{\beta}$	$\alpha\eta\overline{\beta}$	$\overline{\alpha}\eta$	$\overline{\alpha}\beta$	0	0
11	$\overline{\eta}$	$\alpha\eta$	0	0	0	0	0	$\overline{\alpha}\eta$

Table 1: Action Probabilities

which is already cached somewhere else, or on a remote miss on the cache line forming the head of the sharing list of which the observed line forms the tail.

Observe that the transitions $5_r \rightarrow 7_r$ and $6_r \rightarrow 8_r$ refer to the variable D_r. The transition $5b \rightarrow 7b$, for example, occurs when a node not containing a copy of the observed line in its cache (on average there are $K - D_b$ of these) has a read miss on the same shared region variable (of which there are $N - m$ in total) which is cached in the observed line.

The transitions for Region II involve only eleven states as there is no need to distinguish control variables from other data. The transition rates are similar to those of Region I and are thus omitted.

The balance equations for bothe Regions I and II can be derived from the transition rates in the usual way—see for example [10]. These equations are not linear, but define a *fixed point* on the vector of steady-state probabilities for each region and are solved iteratively.

Probabilities of Each Action

With the steady state probabilities now known, we can calculate the probability that a processor emerging from a think state generates each action (the probability distribution of the state at such instants is the same since the memory access stream is assumed Poisson).

For the actions including a DELetion, one has to distinguish the cases according to the state of the location, because the same kind and number of messages is not sent, as will be shown in the next section. The probability of each action occurring in each state in Region I is given in Table 1. Note that the notation $\overline{x}$ denotes $1 - x$ and that η is an aggregate of η_a and η_b from the transition rates, given by:

$$\eta = \frac{(q_{3_a} + q_{3_b} + q_{4_a} + q_{4_b} + q_{5_a} + q_{5_b} + q_{6_a} + q_{6_b})\ n\ (K-1)}{N}$$

We will denote this table by δ so that, for example, $\delta_{7,DC} = \overline{\eta}\overline{\beta}$. The equivalent table for Region II will be denoted δ' and is the same as Table 1 except that η' replaces η throughout.

3.3 Message Streams

In order to calculate the time these messages take to be transmitted we need to calculate the rate at which they are produced. Table 2 shows, for each action and state, how many short and long messages are sent, and how many cache and memory accesses are involved in satisfying a memory request and in maintaining the sharing list(s) concerned. We will assume that the message assembly times are negligible in comparison to the ring transmission and cache/memory access times. Those messages which involve communication with memory are marked with a '*';

these messages which may by-pass the ring if the required data is held locally. As already stated, shareable data is assumed to be evenly distributed across the component nodes of the machine whilst private data is more likely to be held local to the initiating processor. In the table, therefore, the term x^* can be read as $x(K-1)/K$.

Note that reductions are a special case since the reducing processor sends and receives as many short messages as there are members in the sharing list behind its cached block. We estimate this by using the mean length of a non-singleton sharing list within each region. We require five values here depending on whether or not the writer is already a member of the sharing list to be reduced. Δ'_H denotes the mean length of a sharing list of which the writer is already a part. Δ'_M denotes the mean length of a sharing list prior to the writer adding itself as a result of a write miss. Similar quantities are defined for Region I appropriately annotated with the line type (a or b). Note that when joining a sharing list in Region I for the purposes of reduction we cannot determine which line type we are joining. We therefore define Δ_M to be a weighted average of Δ^a_M and Δ^b_M in this case. Hence,

$$\begin{aligned}
\Delta'_H &= D' \\
\Delta'_M &= P'_{ex} + P'_{sh} E[\, L' \mid L' \neq K \,] \\
&= P'_{ex} + P'_{sh} \frac{\sum_{l=1}^{K-1} l P'_l}{1 - P'_K} \\
\Delta^r_H &= D \\
\Delta^r_M &= P_{ex} + P_{sh} + \frac{\sum_{l=1}^{K-1} l P_l}{1 - P^r_K} \\
\Delta_M &= \frac{\gamma_1 \Delta^a_M + \gamma_2 \Delta^b_M}{\gamma_1 + \gamma_2}
\end{aligned}$$

where $r = a, b$ again denotes the line type. The various P_{ex} and P_{sh} terms denote the probability of a cached copy of a line being exclusive and shared respectively. For example, P'_{ex} is the probability that a shareable line is cached in an exclusive state in Region II, i.e.

$$P'_{ex} = \frac{q'_3 + q'_4}{q'_3 + q'_4 + q'_5 + q'_6}$$

with $P'_{sh} = 1 - P'_{ex}$. The same quantities for Region I follow similarly.

As an example, an AD action first experiences a cache miss and then takes a copy of the required block, adding it to the head of the current sharing list. It thus issues one short message to memory requesting the head of the sharing list (involving a memory access); this in turn sends a short message to the head of the list (involving a cache access), which in turn sends a long message containing the latest copy of the line to the originating processor (this must be written to the processor's cache and therefore involves one further cache access). Either one of the short messages to/from memory may be local, hence the 1^* in the table. Note that the three Delete actions in states 2 and 4 each cause a write back (via a long message) of the dirty line to memory.

This table implicitly defines the non-zero entries of six new tables S, S', L, C, C' and M representing the number of short messages (Region I and Region II respectively), long messages, cache accesses (Region I and Region II respectively) and memory accesses, indexed by the line state and the action initiated. Thus, for example, $S'_{6,DA} = 2^*+2$, $C_{9_a,TR} = 2+\Delta^a_H$ and $L_{3,AD} = 0$. The table is used to determine the *total* message traffic over the ring. However, in computing the mean memory access time, we will need to take into account the fact that some messages can be sent in parallel so that not all of the associated message delays associated with each action will contribute to the total time taken to complete the action. This therefore requires a separate table describing the actual delay associated with each action. In order to construct this table we assume that the time taken to construct a message is negligable in comparison to the time taken to transmit it. Thus, if two messages of equal length are required to be transmitted simultaneously

Action	*State*	*Short*		*Long*	*Cache*		*Memory*
		Region I	Region II		Region I	Region II	
		state (superscript r) (S)	(S')	(L)	state (superscript r) (C)	(C')	(M)
CR	11	1^*	1^*	1^*	2	2	1
AD	11	1^*+1	1^*+1	1	3	3	1
DC	1	1^*	1^*	1^*	2	2	1
	2	1^*	1^*	1^*+1	2	2	2
	3	2^*	2^*	1^*	2	2	2
	4	1^*	1^*	2^*	2	2	2
	5,6	2^*+1	2^*+1	1^*	3	3	2
	7,8	1^*+2	1^*+2	1^*	4	4	1
	9,10	1^*+1	1^*+1	1^*	3	3	1
DA	1	1^*+1	1^*+1	1	3	3	1
	2	1^*+1	1^*+1	2	3	3	2
	3	2^*+1	2^*+1	1	3	3	2
	4	1^*+1	1^*+1	1^*+1	3	3	2
	5,6	2^*+2	2^*+2	1	4	4	2
	7,8	1^*+3	1^*+3	1	5	5	1
	9,10	1^*+2	1^*+2	1	4	4	1
DR	3	2^*+1+2Δ_M	2^*+1+2Δ'_M	1	3+Δ_M	3+Δ'_M	2
	4	1^*+1+2Δ_M	1^*+1+2Δ'_M	1^*+1	3+Δ_M	3+Δ'_M	2
	5,6	2^*+2+2Δ_M	2^*+2+2Δ'_M	1	4+Δ_M	4+Δ'_M	2
	7,8	1^*+3+2Δ_M	1^*+3+2Δ'_M	1	5+Δ_M	5+Δ'_M	1
	9,10	1^*+2+2Δ_M	1^*+2+2Δ'_M	1	4+Δ_M	4+Δ'_M	1
TR	7, 8	1^*+3+2($\Delta^r_H - 1$)	1^*+3+2($\Delta'_H - 1$)	0	3+Δ^r_H	3+Δ'_H	1
	9, 10	1^*+2+2($\Delta^r_H - 1$)	1^*+2+2($\Delta'_H - 1$)	0	2+Δ^r_H	2+Δ'_H	1
HR	5, 6	2($\Delta^r_H - 1$)	2($\Delta'_H - 1$)	0	Δ^r_H	Δ'_H	0
IR	11	1^*+1+2Δ_M	1^*+1+2Δ'_M	1	3+Δ_M	3+Δ'_M	1

Table 2: Message, cache and memory traffic for each action/state pair

Action	*Short*		*Long*	*Cache*		*Memory*
	Region I (state subscript r) ($\overline{\overline{S}}$)	Region II ($\overline{\overline{S'}}$)	($\overline{\overline{L}}$)	Region I (state subscript r) ($\overline{\overline{C}}$)	Region II ($\overline{\overline{C'}}$)	($\overline{\overline{M}}$)
CR	1^*	1^*	1^*	2	2	1
AD	1^*+1	1^*+1	1	3	3	1
DC	1^*	1^*	1^*	2	2	1
DA	1^*+1	1^*+1	1	3	3	1
DR	1^*+1+2Δ_M	1^*+1+2Δ'_M	1	3+Δ_M	3+Δ'_M	1
TR	1^*+1+2Δ^r_H	1^*+1+2Δ'_H	0	3+Δ^r_H	3+Δ'_H	1
HR	2($\Delta^r_H - 1$)	2($\Delta'_H - 1$)	0	Δ^r_H	Δ'_H	0
IR	1^*+1+2Δ_M	1^*+1+2Δ'_M	1	3+Δ_M	3+Δ'_M	1

Table 3: Actual message, cache and memory delays incurred

to complete an action then the delay associated with message transmission will be taken to be the time taken to send just one of the messages.

We will distinguish these tables from those above by means of a double overbar on the table name, e.g. $\overline{\overline{S}}$. These are summarised in Table 3.

The rate of generation of short and long messages to the ring, λ_s and λ_l say, are now given by:

$$\lambda_s = \tau(P_I \sum_{s,a} q_s \ \delta_{s,a} \ S_{s,a} + P_{II} \sum_{s',a} q'_{s'} \ \delta'_{s',a} \ S'_{s',a})$$

$$\lambda_l = \tau(P_I \sum_{s,a} q_s \ \delta_{s,a} \ L_{s,a} + P_{II} \sum_{s',a} q'_{s'} \ \delta'_{s',a} \ L_{s',a})$$

where $\tau = \pi\tau_0$ is the rate at which memory requests are submitted by a processor, and where P_I and P_{II} represent the probability that a memory reference addresses the cache in Regions I and II respectively:

$$P_I = \gamma_1 + (\gamma_2 + \gamma_3(1-\sigma))\frac{m}{n}$$
$$P_{II} = (\gamma_2 + \gamma_3(1-\sigma))\frac{n-m}{n}$$

Note that $P_I + P_{II} = 1 - \gamma_3\sigma$.

3.4 Mean transmission time for a message

We now determine the mean transmission time of a message around the ring. For simplicity we make the assumption that all echo messages have the same length as a short message. Thus a short message issued by a transmitting node will perform one full circuit of the ring (i.e. through $K-1$ ring buffers). The receiving node will extract the incoming packet and, at the same time, pass it on as an echo packet. A long message will be converted to a short echo message once it has reached the receiving node.

It is important to remember that messages originating from the ring have priority over messages in the transmit queue originating from the node. In order to model these priorities we will use Cobham's formulae [10] to determine the mean waiting times of messages in the transmit queue and ring buffer, W_t and W_r respectively. From these we can obtain the mean transmission time of short and long messages around the ring, T_s and T_l respectively.

To use Cobham's formulae we need to determine the rate at which messages are generated by the transmit queue and ring buffer of each node. We will denote these by λ_t and λ_r respectively. The total rate at which messages are generated by a node is thus:

$$\lambda_t = \lambda_s + \lambda_l$$

Suppose the length of a long message is a fixed multiple M of the length of a short message. The various transmission times and utilisations can then be expressed in terms of the transmission time of a short message, which we denote by t_{short}.

The proportion of short and long messages within the aggregated message stream from each node, p_s and p_l respectively, can be written in terms of the corresponding arrival rates of long and short messages:

$$\begin{aligned} p_s &= \frac{\lambda_s}{\lambda_s + \lambda_l} \\ p_l &= \frac{\lambda_l}{\lambda_s + \lambda_l} \end{aligned}$$

To determine the corresponding proportions, p'_s and p'_l, for messages emanating from the ring we must take into account the fact that long messages are converted to (short) echo messages once they have reached their destination. Thus, a short message circulates through $K-1$ ring buffers and each long message through an average of $(K-2)/2$ ring buffers before being converted to a (short) echo packet. This echo packet is circulated through the remaining $K/2$ buffers back to the transmitter. Hence, ring traffic has total rate

$$\lambda_r = (K-1)\ \lambda_s + (K/2)\ \lambda_l + ((K-2)/2)\ \lambda_l$$

and

$$\begin{aligned} p'_s &= \frac{(K-1)\ \lambda_s + (K/2)\ \lambda_l}{\lambda_r} \\ p'_l &= \frac{((K-2)/2)\ \lambda_l}{\lambda_r} \end{aligned}$$

If we define $X \in \{1, M\}$ to be the number of short message equivalents in each message in the aggregate stream emanating from each node then the n^{th} moment of X is given by:

$$M_n = p_s + M^n p_l$$

Similarly for the number of short message equivalents in each message from the ring we obtain the n^{th} moment as:

$$M'_n = p'_s + M^n p'_l$$

We use these to derive the mean waiting times below.

In order now to use Cobham's formulae we shall make the assumption that the stream of messages arriving at the ring buffer are *Poisson*. They are not in general, but the assumption is common and often found to be reasonable, especially in representing superposed traffic.

If t_{short} is the time taken for a link to service the transmission of a short message then the mean message service time is $M_1\, t_{short}$ and we define:

$$\begin{aligned} \rho_t &= \lambda_t M_1 t_{short} \\ &= \lambda_t(\lambda_s/\lambda_t + M\lambda_l/\lambda_t)t_{short} \\ &= t_{short}(\lambda_s + M\ \lambda_l) \end{aligned}$$

So we define

$$\begin{aligned} \rho_r &= \lambda_r M'_1 t_{short} \\ &= t_{short}((K-1)\ \lambda_s + (K/2)\ \lambda_l + M((K-2)/2)\ \lambda_l) \end{aligned}$$

We can now obtain the queueing times of messages in the transmit buffer and ring buffer respectively:

$$\begin{aligned} Q_t &= \frac{W_{0t}}{(1-\rho_r)(1-\rho_r-\rho_t)} \\ Q_r &= \frac{W_{0r}}{1-\rho_r} \end{aligned}$$

where

$$\begin{aligned} W_{0r} &= \frac{1}{2} t_{short}^2 \; M_2 \; \lambda_t \\ W_{0t} &= \frac{1}{2}(t_{short}^2 \; M_2 \; \lambda_t + t_{short}^2 \; M_2' \; \lambda_r) \\ &= \frac{1}{2} t_{short}^2 [(\lambda_s + \lambda_l \; M^2) + ((K-1)\lambda_s + (K/2) \; \lambda_l \\ &\quad +((K-2)/2) \; \lambda_l M^2)] \end{aligned}$$

¿From this we can now determine the mean transmission time of short and long messages to be:

$$\begin{aligned} T_s &= K t_{short} + Q_t + (K-1) Q_r \\ T_l &= (K/2)(1+M) t_{short} + Q_t + (K-1) Q_r \end{aligned}$$

In order to derive the mean time to service a memory request, T we must incorporate the times for all the required cache and memory accesses. In order to account for the memory/cache access delay, we need to consider the cache/memory controller which receives independent request streams from the node processor(s) and the SCI ring.

3.5 Cache/Memory Access Delay

We will assume that memory request streams from the processor and the ring are given equal priority so that queueing is on a first-come-first-served basis.

As the memory and cache access times are constant we utilise a separate M/G/1 model of the cache/memory controller in which the service time distribution is a probabilistic mixture of the two. We shall refer to the individual times as t_{cache} and t_{mem} respectively.

In order to demonstrate the application of the model we will use it to compare the performance of two possible node configurations: the first is the standard configuration shown in Figure 1 and the second an alternative design (Figure 2) in which the node memory is located on the system bus. The second design is interesting to compare with the standard design, as the system bus architecture (indicated by the dashed box) in the latter is simply that of a standard shared-memory multiprocessor. The idea here is to investigate the suitability of SCI as the interconnect medium for building extensible parallel machines from existing bus-based multiprocessors. We consider the revised design in more detail below, but begin with the configuration shown in Figure 1.

Standard Memory Model

In the standard memory model all SCI memory requests are handle by a single cache/memory controller and in the model this is represented by a single shared queue. The requests are generated both by the processor(s) and the SCI ring and contain a mixture of cache and memory requests. We distinguish the two in the model which allows changes in the ratio of cache speed to memory speed to be explored.

We now determine the mean number of cache and memory accesses (n_c and n_m respectively) for each action in a similar manner to the number of short and long messages:

$$\begin{aligned} n_c &= P_I \sum_{s,a} q_s \; \delta_{s,a} \; C_{s,a} + P_{II} \sum_{s',a} q_s' \; \delta_{s',a}' \; C_{s',a}' \\ n_m &= P_I \sum_{s,a} q_s \; \delta_{s,a} \; M_{s,a} + P_{II} \sum_{s',a} q_s' \; \delta_{s',a}' \; M_{s',a}' \end{aligned}$$

For the purposes of calculating the memory access delay, it is convenient to define similar terms $\overline{\overline{n_c}}$ and $\overline{\overline{n_m}}$ in which $\overline{\overline{C}}, \overline{\overline{C'}}, \overline{\overline{M}}$ and $\overline{\overline{M'}}$ are used instead of C, C', M and M' respectively. These adjusted rates take into account the fact that some messages can be sent in parallel as described above. Similarly we define $\overline{\overline{n_s}}$ and $\overline{\overline{n_l}}$ to denote the mean number of short and long messages respectively for each action.

From these we can calculate the rate at which cache and memory accesses are produced by a processor (λ_c and λ_m repectively):

$$\begin{aligned}
\lambda_c &= \pi\tau(1-\gamma_3\sigma) + \lambda_t \frac{n_c}{n_c+n_m} \\
\lambda_m &= \pi\tau\gamma_3\sigma + \frac{\pi\tau(\gamma_1+\gamma_2)(1-\beta)}{K} + \lambda_t \frac{n_m}{n_c+n_m}
\end{aligned}$$

Since the total arrival rate of cache and memory accesses is $\lambda_{cm} = \lambda_c + \lambda_m$ we finally obtain the mean queuing time at the cache/memory controller from the Pollaczek-Khinchine formula:

$$\begin{aligned}
Q_{cm} &= \frac{M_2\lambda_{cm}}{2(1-\rho_{cm})} \\
&= \frac{\lambda_{cm}}{2(1-\rho_{cm}} \left(\frac{\lambda_c t_{cache}^2}{\lambda_c+\lambda_m} + \frac{\lambda_m t_{mem}^2}{\lambda_c+\lambda_m} \right) \\
&= \frac{\lambda_c t_{cache}^2 + \lambda_m t_{mem}^2}{2(1-\rho_{cm})}
\end{aligned}$$

where ρ_{cm} is the cache/memory controller utilisation given by:

$$\rho_{cm} = \lambda_c t_{cache} + \lambda_m t_{mem}$$

¿From this we can now determine the mean time to service a memory request:

$$\begin{aligned}
T(\pi) &= (1-\sigma\gamma_3)\left(\overline{\overline{n_s}}\,T_s + \overline{\overline{n_l}}\,T_l + \overline{\overline{n_c}}(Q_{cm}+t_{cache}) + \overline{\overline{n_m}}(Q_{cm}+t_{mem}) + p_{hit}\,(Q_{cm}+t_{cache})\right) \\
&+ \sigma\gamma_3(Q_{cm}+t_{mem})
\end{aligned}$$

remembering that local private accesses are serviced directly by the memory, cache hits are serviced by the local cache and cache misses by the SCI protocol. p_{hit} here is the probability of a cache hit and is given by:

$$p_{hit} = \frac{P_I}{P_I+P_{II}} \left(1 - \sum_{s,a} \delta_{s,a} \right) + \frac{P_{II}}{P_I+P_{II}} \left(1 - \sum_{s',a} \delta'_{s',a} \right)$$

We are now in a position to determine the processor utilisation π. The model here is open so that a processor may submit new memory requests before previous requests have been satisfied. However, we make the assumption that a processor will stall if the number of outstanding memory requests exceeds some constant ω, provided as a parameter to the model. The mean memory response time, as determined above, is given by $T(\pi)$ so that π is the probability that fewer than ω memory requests are submitted in time $T(\pi)$, that is

$$\pi = 1 - \sum_{i=1}^{\omega-1} \frac{(\tau T(\pi))^i e^{-\tau T(\pi)}}{i!}$$

since the memory request arrival stream is Poisson. Because of the mutual depency of $T(\pi)$ and π, the above calculation in fact produces a refinement to some previous estimate of π. $T(\pi)$ and π must therefore be repeatedly recalculated until convergence is achieved.

Note that if the memory requests are produced by a collection of P statistically identical independent processors, as in Figure 1, the number of active processors at any time is Binomially

distributed with parameters P and π. The mean is therefore $P\pi$ so that messages are generated at net rate $P\pi\tau_0$ where τ_0 is the message generation rate for each processor. Thus, for P processors,

$$\pi = 1 - \sum_{i=1}^{P\omega-1} \frac{(P\tau T)^i e^{-P\tau T}}{i!}$$

Revised Memory Model

We now consider the configuration shown in Figure 2. The operation of this design is similar to that of the original except that requests to the memory from the SCI ring may now have to compete with other requests on the local system bus. Equally, however, some accesses to memory, specifically private accesses which require no intervention from the SCI cache, can proceed independently of the SCI node controller. Thus the relative performance of the two designs is determined by the nature of the workload.

Requests from the SCI ring will be diverted to the system bus if the request is a read to a home block which has been cached and dirtied by a local processor, or if the request is an invalidation (i.e. part of a list reduction). In order to estimate the memory access time, and also the load on the system bus, we need an additional workload parameter in this case, which is the system bus load due to local (i.e. snoopy protocol) coherency traffic among the processor/cache units. We shall refer to this as τ'.

The traffic on the bus comes from either the processors attached to it, or from the ring. The total arrival rate to the bus is therefore:

$$\lambda'_{bus} = \pi\tau + \tau' + \lambda_t \frac{n_m}{n_c + n_m}$$

The queueing time is then given by:

$$Q'_{bus} = \frac{\lambda'_{bus}\, M_{2_{bus}}}{2(1-\rho'_{bus})}$$

The rate at which the cache/memory directory supplies data to the system bus, τ_{cm} say, is given by

$$\tau_{cm} = \pi\tau(1 - \gamma_3\sigma - \frac{\gamma_1+\gamma_2}{K}(1-\beta))$$

so that:

$$\begin{aligned} \rho'_{bus} &= \tau_{cm} t_{cache} + (\lambda'_{bus} - \tau_{cm}) t_{mem} \\ M_{2_{bus}} &= \frac{\tau_{cm}}{\lambda'_{bus}} t^2_{cache} + (1 - \frac{\tau_{cm}}{\lambda'_{bus}}) t^2_{mem} \end{aligned}$$

The same quantities for the cache-memory controller bus are then given by:

$$\begin{aligned} \lambda'_{cm} &= \pi\tau(1-\gamma_3\sigma) + \lambda_t \\ Q'_{cm} &= \frac{\lambda'_{cm} t^2_{cache}}{2(1-\rho'_{cm})} \\ \rho'_{cm} &= \lambda'_{cm} t_{cache} \end{aligned}$$

This concludes the model.

4 Numerical Results

We now consider an SCI configuration with four 20 MIPS processors at each node, and compare the performance of the two memory configurations under varying workloads. We assume that

memory is accessed at the rate of one reference for every 5 instructions executed and that the hit rate at the processors private cache (not the SCI cache) is 95%. From this we can estimate τ as $2 \times 10^7 \times 0.2 \times 0.05 = 2 \times 10^5$. It should be appreciated that these are only sample parameterisations and, although reasonable in comparison with some hardware configurations, they are not meant to represent any particular machine.

The example here assumes an SCI ring with a varying number of nodes. We assume a total working set of 10^6 memory blocks, with an SCI cache size of 10^3 lines per node, and 10^2 shared control variable blocks in the application code. The read probability is $\alpha = 0.7$ and the shared region access probability is taken to be $\gamma_2 = 0.2$. The other time parameters assumed are $t_{cache} = 240ns, t_{mem} = 480ns$ (note that reading and writing a line involves several memory cycles). The SCI ring is assumed to be fibre optic and is taken to operate at 1Gbit/sec so that a short message of 16 bytes takes $t_s = 64ns$ to transmit from one ring node to the next. A long message is taken to be 80 bytes on average, i.e. 5 times that of a short message. The additional coherency traffic on the system bus is taken to be the equivalent of 10

The graphs shown below are produced by an implementation of the model written in Mathematica [12].

To illustrate the performance trends as the number of nodes (K), hit rate (β) and control variable access probability (γ_1) are varied, Figure 3 shows the *elongation* when 10% of all memory accesses are to control variables (in cache Region I), i.e. when $\gamma_1 = 0.1$, for $\beta = 0.8, 0.9, 0.95$ and 0.98. The elongation is a measure of the overhead on memory access time that is imposed by SCI. It is calculated as the ratio of the actual memory service time to that of an "ideal" system in which all cache misses are serviced in exactly one memory access time. That is:

$$elongation = \frac{T}{\beta\, W_{cache} + (1-\beta)\, W_{mem}}$$

The system scales reasonably well up to 16 processors, but then degrades rapidly as the ring saturates. The steep curve for $\beta = 0.8$ beyond this point is due to long queues building up at the node transmitters and cache/memory controllers because of the increased ring traffic needed to service cache misses. In practice, the number of outstanding memory requests will be limited by the degree of run-on which each processor can achieve after issuing a memory request.

A similar graph for a fixed value of $\sigma = 0.95$ is shown in Figure 4 for comparison. The uniform case shows higher contention, the processor utilisation dropping off more quickly with a corresponding increase in elongation.

To give a direct comparison between the two node architectures we show in Figures 5 and 6 the elongation curves for $\sigma = 0.95$ and $\beta = 0.95$ with $\gamma_1 = 0.1$ and 0.3 in combination. Using the same value of τ_0 as above, there is no perceivable difference between the performance of the two node architectures. We therefore use $\tau_0 = 2 \times 10^6$ here to demonstrate a crossover point between the two models.

An analysis of the memory (i.e. bus) queues reveals that the critical factor affecting the models is contention for access to the node memory. Recall that in the revised model memory accesses from remote processors have to contend with local system bus traffic, including local coherency protocols. Equally, however, remote accesses to the SCI cache can now be serviced at the same time as accesses to the local memory. This leads to a tradeoff between the two designs with the trends being hard to predict in the absence of a model.

5 Summary

In this paper we have developed a new analytical model of the IEEE Standard Coherent Interface running on a unidirectional ring architecture. We have essentially combined two existing modelling approaches: one for bus-based shared-memory systems in which the processor utilisation is derived indirectly from the equilibrium cache line state probabilities, and the other for analysing slotted rings using an underlying M/G/1 queueing model. However, we have also explicitly modelled the

cache/memory system within each node and have built in a non-uniform memory reference model in order to better approximate the patterns of memory referencing found in real systems.

We have presened some sample numerical results for the processor utilisation and SCI overheads, for a particular parameterisation of the system, as the number of processors is increased. This demonstrates the expected eventual saturation of the SCI ring as the number of processors is increased, and as the cache hit rates are reduced. We have demonstrated the application of the model by considering two alternative node architectures and by analysing the tradeoffs between the two designs. This has exposed some non-intuitive behaviour regarding traffic flows to the node memory and cache units—effects which would be hard to predict in the absence of a performance model.

5.1 Future work

Model validation has not been possible to date because neither a real SCI system nor a suitable execution-driven simulator for SCI has been available. Detailed simulation models do exist for other distributed coherency protocols, however, and currently we are validating a model of the protocol defined in [17] which has been developed using the same approach as has been used here. This can be expected to provide general insights into the accuracy of this type of model. Subsequently we intend to undertake a full validation exercise against early hardware implementations as they become available.

In the current model the processors at each node on an SCI ring are modelled as a single resource which generates read and write requests to the memory system via a system bus. In practice, however, the system bus is attached to a number of separate processors, each with a local private cache and typically operating a "snoopy"style coherency protocol on the bus, independently of the SCI system. Thus, when modelling this bus, the additional bus load caused by this coherency traffic is represented by a single additional parameter which is estimated based on results from earlier papers such as [7]. It would be possible to develop a sub-model of this bus system in order to caclulate the additional coherency overhead directly in terms of our existing workload parameters. This will provide a more accurate and flexible model of a multiple shared-memory multiprocessor architecture, in keeping with the initial objectives.

A further important enhancement is one which will enable the node memory blocks to be considered as cache lines. In the present model the cache and memory are separate entities so that if a memory block is requested by one of the processors local to its node the line is assumed to be transferred to the node cache. In the physical hardware, additional line state information may be associated with each memory block to obviate the need for this memory to cache transfer. This involves a substantial complication of the central Markov model since the node memory must now be considered part of the node cache. This involves introducing line states for memory locations and allowing shared memory locations to acquire sharing list status when requested by remote processors.

SCI is an extremely complicated standard and there are many features which have been ignored in the model. Transmission failures and retries may prove to be important components of the protocol which we have not yet considered. Also, flow control, which provides greater control over the transmission rates of each processor, is also a significant feature which the model ignores. We could explore an equivalent closed model of the system in which each processor is limited to a certain number of outstanding memory requests. In practice, a processor does not always stall when it issues an address; instead it will continue to execute instructions in the pipeline until one is blocked by an outstanding request. It is therefore important to ensure that the request population in the system is set at a realistic value to model reasonably accurately processor run-ons of this type. In the mean time we have settled for an open model. This is less accurate for low hit rates, and high displacement traffic because of the large number of outstanding references which the open model will generate. However, this is a bad combination in any machine and at least the open model will be over-pessimistic in its estimate of the memory response time.

As stated in the introduction, SCI is not restricted to a ring and an obvious extension of this work is to use the Markov model for the SCI cache states within a more sophisticated communi-

cation network model. The rapid saturation of the ring as the number of nodes increases is no surprise in this case. The significant contribution of this paper, however, is the development of the model for the SCI coherency traffic in the non-uniform memory reference case and it will be interesting to see how well SCI scales when this is incorporated into models for more elaborate networks such as meshes, crossbars and hypercubes.

References

[1] A. Agarwal, R Simoni, J Hennessy and M Horowitz. "An evaluation of directory schemes for cache coherence". *In Proc. of the 15th Annual Int. Sym. on Computer Architecture*, p 280-289, June 1988

[2] J.W. Bothner and T.I. Hulaas, *Various Interconnects for SCI-based Systems*

[3] L. Censier and P. Feutrier. "A new solution to coherence problems in multicache systems". *IEEE Trans. on Computers*, c-27(12):112-118 December 1978.

[4] A. Gupta, W-D Weber and T. Mowry, *Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Systems*

[5] S. Gjessing, D.B. Gustavson, J.R. Goodman, D.V. James, E.H Kristiansen. "The SCI Cache Coherence Protocol". In *Scalable Shared Memory Multiprocessors*, M. Dubois and S. Thakkar, eds., Kluwer academic Publishers, Norwell, Mass. 1992

[6] The IEEE. "IEEE P1596 Standard Specification". IEEE Publication, 1989.

[7] A.G. Greenberg and I.Mitrani "Analysis of Snooping Caches". *Proc. of Performance 87, 12th Int. Symp. on Computer Performance*, Brussels, December 1987.

[8] E. Ametistova and I.Mitrani "Modelling and Evaluation of Cache Coherence Protocols in Multiprocessor Systems". *Proceedings of the 9th U.K. Performance Engineering Workshop, Loughborough, 1993.*

[9] T.M. Cole, Private Communication, 1994.

[10] P.G. Harrison and N.M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley. 1993

[11] D.V. James. "SCI Cache Coherence". In *Cache and Interconnect Architectures in Multiprocessors*, M. Dubois and S. Thakkar, eds., Kluwer academic Publishers, Norwell, Mass. 1990.

[12] Wolfram Research. *Mathematica - A System for Doing Mathematics by Computer*, Wolfram research, 1994.

[13] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta and J. Hennessy. "The Directory-based Cache Coherence Protocol for the DASH Multiprocessor". sym on Computer Architecture. *In Proc. of the 17th Annual Int. Sym. on Computer Architecture.* June 1990

[14] S.L. Scott, J.R. Goodman and M.K. Vernon. "Performance of the SCI ring". *In Proc. of the 19th Annual Int. Sym. on Computer Architecture.* May 1992.

[15] B. Spiers and H. Wiggers. "Performance Analysis of SCI Cache Coherency Protocol". IEEE P1956 doc124. 1989.

[16] R. Hexsel and N. Topham, "Performance of SCI Memory Hierarchies", In Proeceedings of 8th International Workshop on Support for Large-Scale Memory Architectures, April 1994.

[17] A. Saulsbury, T. Wilkinson, J. Carter, and A. Landin, "An Argument For Simple COMA", First IEEE Symposium on High Performance Computer Architecture, Rayleigh, North Carolina, USA, pp. 276-285, January 1995.

[18] J.P. Singh, W-D Weber and A. Goopta, "Stanford Parallel Applications for Shared Memory", Technical Report No. CSL-TR-91-469, Stanford Systems Laboratory, Stanford 1991.

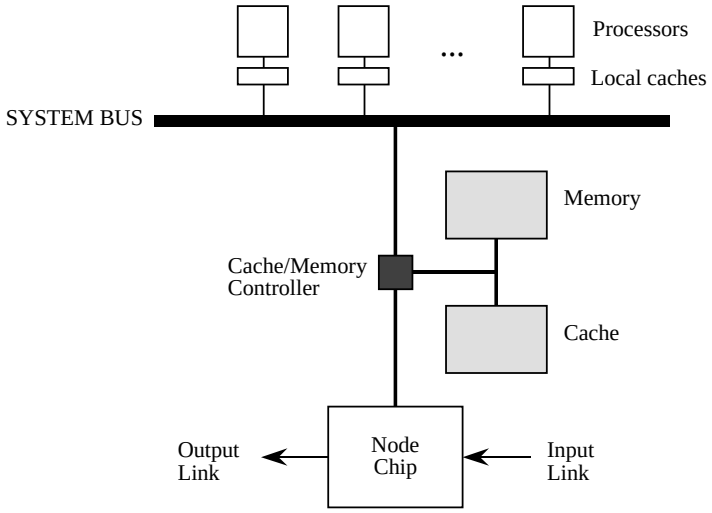

Figure 1: Basic Node Architecture

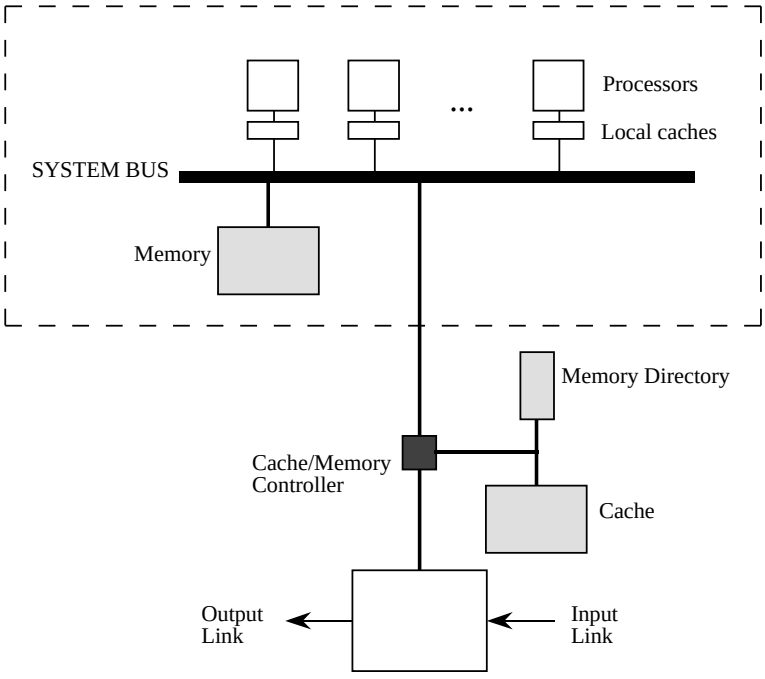

Figure 2: Revised Node Architecture

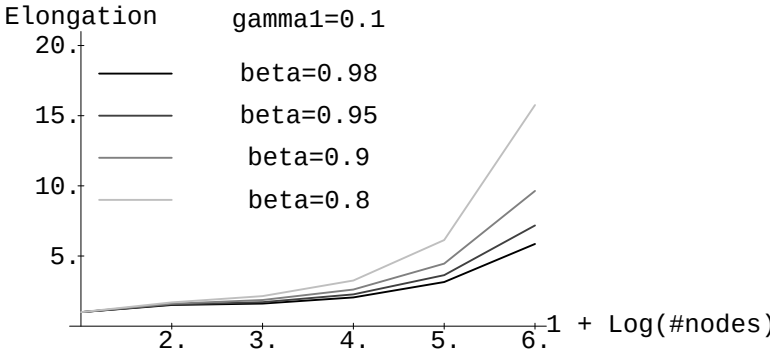

Figure 3: Elongation as a function of K $(\gamma_1 = 0.1, \sigma = 1/K)$

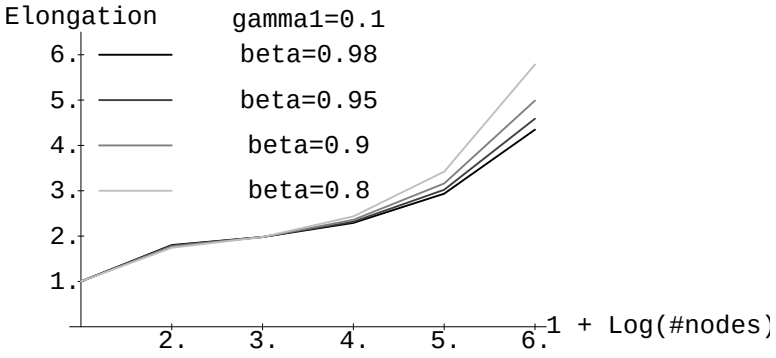

Figure 4: Elongation as a function of K $(\gamma_1 = 0.1, \sigma = 0.95)$

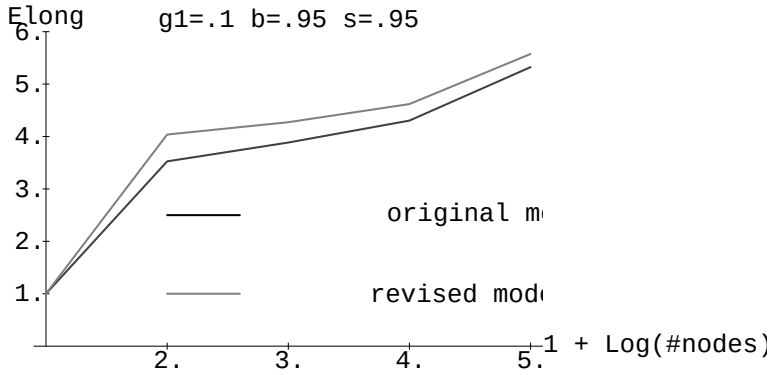

Figure 5: Comparison of Elongations $(\gamma_1 = 0.3, \beta = 0.95, \sigma = 0.95)$

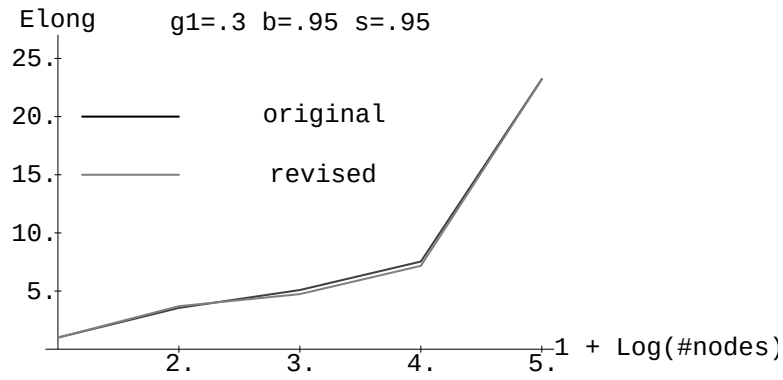

Figure 6: Comparison of Elongations ($\gamma_1 = 0.3, \beta = 0.95, \sigma = 0.95$)