# A Fixed-Point Model of a Distributed Memory Consistency Protocol

A.J. Field and P.G. Harrison

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ U.K.

**Summary.** *We present a new analytical performance model of the IEEE P1596 Standard Coherent Interface operating on the default unidirectional ring architecture. The performance metrics are derived from the equilibrium probability of a cache line being in a given state; these are found by solving a set of fixed point equations. From this we derive expressions for the message traffic emanating from each node and over the ring taking into account the relevant traffic priorities in SCI. Further analysis then yields the mean memory access time and processor utilisation. We demonstrate the application of the model by comparing the performance of two different node configurations.*

## 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) [3] and we assume an implementation of the protocol on the default unidirectional ring network. Full details of the protocol may be found in [2, 3], for example, but we note the key point which is that, in the absence of a shared bus for broadcasting coherency information, the sharers of a cache line are represented explicitly in the form of a *sharing list*. 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.

To date, the only analytical model relevant to SCI that we are aware of is that of [5] 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 SCI cache coherency, however, and so does not model the message traffic generated by each node to maintain a coherent global memory. Our approach is based on that of [4] for modelling bus-based coherency protocols; it differs in its characterisation of memory usage and in the way it models ring traffic and queueing for the cache and memory.

### 1.1 Workload Assumptions

We assume that memory accesses are 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 relatively small in number but addressed with relatively high frequency. The shared areas contain larger data structures, typically with access priveliges governed by control variables, that are relatively infrequently accessed. It is assumed that the control variables are memory mapped in such a way that they may *all* reside in the SCI cache at the same time so that there are no conflicts in the cache between accesses to these variables. The cache thus contains disjoint sets of lines: those which may contain both control variables and shared area data (we call this "Region I" of the cahe) and those which may contain only shared area data ("Region II"). Within the control variables and shared areas, memory accesses are assumed to be uniformly distributed; within each region, the cache lines are then statistically identical.

## 2. A New Model for SCI

The approach we have taken uses the basic method in [4] to obtain the equilibrium cache line state probabilities and uses them to determine the message arrival rates for a separate model of the ring.

We assume the following parameters:

$\alpha$ – the probability of a read
$\beta$ – the probability of a hit in the SCI cache
$\gamma_1$ – the probability that a memory access is to a control structure
$\gamma_2$ – the probability that a memory access is to the shared areas
$\gamma_3$ – the probability that a memory accesses is to private data
$\sigma$ – the probability that an access to private data, which is mapped into the global address space, is held locally
$K$ – The number of nodes
$N$ – The total number of shareable memory blocks
$m$ – The number of (shareable) lines containing control variables
$n$ – The capacity (in blocks) of each SCI cache
$\tau$ – The rate at which a busy processor leaves the "think" state (note that $1/\tau$ is the mean think period)

Note that $\gamma_1 + \gamma_2 + \gamma_3 = 1$. We aim to determine the probability, $\pi$, that a processor is busy doing useful work and also the mean time to service a memory request.

### Cache Line States

A memory block is either **Home** (uncached) or **Cached** and cached copies may either be *clean* or *dirty*. There are eleven basic line states although eight of them we will subdivide further in accordance with the type of data they

contain (control variables or shared area data). The states are: **1 Private Clean**–The location contains a clean copy of private (non-shareable) data; **2 Private Dirty**; **3 Only Clean**–The location contains the only cached copy of a memory block and is clean; **4 Only Dirty**; **5 Head Clean**–As 3 but it is at the head of a list containing at least two members; **6 Head Dirty**; **7 Mid Clean**–The location is in the middle of the list (i.e. at neither the head nor tail) and the cached copy is clean; **8 Mid Dirty**; **9 Tail Clean**–As 5 but the location is at the tail of the sharing list; **10 Tail Dirty**; **11 Invalid**. For Region I of the cache we 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 which is easily ensured by a suitable allocation of global memory addresses. The cache lines in Region I therefore have nineteen possible states, and those of Region II eleven.

### 2.1 Actions generated by a processor

A given processor emerges from the think state at rate $\tau$, and according to the state of the cache location it is accessing will generate one of the following processor *actions* (the distinguishing subscripts '$a$' and '$b$' are omitted as the actions are the same for each): **Creation (CR)**–A read or write miss on a location in state 11 not cached by other processors; **Addition (AD)**–As above but already cached. **Deletion-Creation (DC)**–A read or write miss on a *home* block mapping to a cache location previously in state 3–10; **Deletion-Addition (DA)**–A read miss on an already *cached* block mapping to a location in state 3–10; **Deletion-Reduction (DR)**–A write miss on a *cached* block mapping to a location in state 3–10; **Take-Reduction (TR)**–A write hit on a block mapping to a location in state 7–10; **Head-Reduction (HR)**–A write hit on a location in the processor's cache but in state 3,4; **Invalid-Reduction (IR)**–A write miss on a *cached* block mapping to a location in state 11.

To add to a sharing list a processor sends a short message to the memory, which sends another short message to the current head, which in turn sends a long message containing the up-to-date copy of the block to the processor. Deleting from a list requires either one or two short messages depending on whether the cache line is at the head/tail or middle of its sharing list. Note that for each Deletion operation in state 2 then a *writeback* of the line will also be performed, requiring one additional long message to the memory.

### 2.2 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, $j = 1, ..., 11$.

### State Transitions

The state transitions occur as a result of read and write operations to the cache. We imagine we are looking at a particular "observed" line of a cache and consider the various read/write operations which can change the state of that line. We omit much of the detail but note that the length of the mid portion of a sharing list is assumed to be geometrically distributed. With this assumption we can easily find the mean sharing list length ($L_a$, $L_b$ for Region I and $L'$ for Region II).

An alternative and apparently more accurate representation of the line states would store the length of each sharing list explicitly with the state in order to avoid assumptions about the distribution of the sharing list length. However, this requires the state of *every* line in the cache system to be considered in order that the correct transitions can be determined. For example, in the event of a read miss the new state of the observed line would be determined by the state (i.e. current length) of the new sharing list to which the line must be added. This cannot be determined by considering the cache line in isolation.

### Transition Rates

The processors leave the think state at rate $\pi\tau$ and we consider those memory accesses which cause a change in line state. 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. Consider Region I first and define $\eta_r$ to be the probability that, given a cache miss on a type $r$ memory block, the block is cached elsewhere and $\epsilon_r$ to be the probability that a cached copy of a type $r$ line is clean ($r$ is either $a$ or $b$). We define:

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

$$\epsilon_r = \frac{q_{3_r} + q_{5_r}}{q_{3_r} + q_{4_r} + q_{5_r} + q_{6_r}}$$

whereupon the transition rates may be written:

$$\{b, 11\} \rightarrow 1 \qquad \frac{\alpha\gamma_3}{n}$$

$$\{1, 2\} \rightarrow 1 \qquad \frac{\alpha(1-\beta)\gamma_3}{n}$$

$$s \rightarrow 2 \qquad \frac{(1-\alpha)\gamma_3}{n}$$

$$\overline{a} \rightarrow 3a \qquad \frac{\alpha\gamma_1(1-\eta_a)}{m}$$

$$\{5a, 9a\} \rightarrow 3a \qquad \frac{p_{3_a}(1-\gamma_1)(1-\beta)}{n}$$

$$\{1, 2, a\} \rightarrow 3b \qquad \frac{\alpha\gamma_2(1-\eta_b)}{n}$$

$$\{b, 11\} \rightarrow 3b, s \neq 5b, 9b \qquad \frac{\alpha(1-\beta)\gamma_2(1-\eta_b)}{n}$$

$$\{5b, 9b\} \rightarrow 3b \qquad \frac{\alpha(1-\beta)\gamma_2(1-\eta_b)}{n} + \frac{p_{3_b}(1-\beta)}{n}$$

$$s \rightarrow 4a, s \neq 6a, 10a \qquad \frac{(1-\alpha)\gamma_1}{m}$$

$$\{6a, 10a\} \rightarrow 4a \qquad \frac{(1-\alpha)\gamma_1}{m} + \frac{p_{3_a}(1-\gamma_1)(1-\beta)}{n}$$

$$s \rightarrow 4b, s \neq 6b, 10b \qquad \frac{(1-\alpha)\gamma_2}{n}$$

$$\{6b, 10b\} \rightarrow 4b \qquad \frac{(1-\alpha)\gamma_2}{n} + \frac{p_{3_b}(1-\beta)}{n}$$

$$\overline{a} \rightarrow 5a \qquad \frac{\alpha\gamma_1\eta_a\epsilon_a}{m}$$

$$7a \rightarrow 5a \qquad \frac{p_{1_a}(1-\gamma_1)(1-\beta)}{n}$$

$$\{1, 2, a\} \rightarrow 5b \qquad \frac{\alpha\gamma_2\eta_b\epsilon_b}{n}$$

$$\{b, 11\} \rightarrow 5b, s \neq 7b \qquad \frac{\alpha(1-\beta)\gamma_2\eta_b\epsilon_b}{n}$$

$$7b \rightarrow 5b \qquad \frac{\alpha(1-\beta)\gamma_2\eta_b\epsilon_b}{n} + \frac{p_{1_b}(1-\beta)}{n}$$

$$\overline{a} \rightarrow 6a \qquad \frac{\alpha\gamma_1\eta_a(1-\epsilon_a)}{m}$$

$$8a \rightarrow 6a \qquad \frac{p_{1_a}(1-\gamma_1)(1-\beta)}{n}$$

$$\{1, 2, a\} \rightarrow 6b \qquad \frac{\alpha\gamma_2\eta_b(1-\epsilon_b)}{n}$$

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

To clarify these equations we consider some specific examples:

- $s \rightarrow 2$, i.e. a transition from any state to the state "Private Dirty". This happens when private data (held in the global address space), whose address maps to the observed cache line, is written to. The associated transition rate is thus $\pi\tau\frac{(1-\alpha)\gamma_3}{n}$.
- $\{1, 2, 3a, ..., 10a\} \rightarrow 6b$. A cache line holding private data or control variables will transit to "head dirty" if a read to a dirty, and already cached, copy of a shared area variable maps to the observed line address. The transition rate is thus $\pi\tau\frac{\alpha\gamma_2\eta_b(1-\epsilon_b)}{n}$.
- $5a \rightarrow 7a, 6a \rightarrow 8a$. This happens on a remote read miss from one of the (average $K - L_a$) processors who do not currently have a cached copy of the control variable in the observed line. A new entry will be added to the sharing list making the observed line state transit from "head" to "mid". The associated rate is $\frac{(K-L_a)\alpha\gamma_1}{m}$.

The transitions for Region II involve only eleven states since they may. In the following equations $\eta$ and $\epsilon$ are the Region II equivalents of $\eta_r$ and $\epsilon_r$, for $r = a, b$ for Region I:

$$
\begin{aligned}
s \rightarrow 1 \quad & \frac{\alpha(1-\beta)\gamma_3}{n} \\
s \rightarrow 2 \quad & \frac{(1-\alpha)\gamma_3}{n}
\end{aligned}
$$

$$
\begin{aligned}
s \to 3, s \neq 9, 5 \quad & \frac{\alpha(1-\beta)(1-\eta)\gamma_2}{n} \\
5 \to 3, 9 \to 3 \quad & \frac{\alpha(1-\beta)(1-\eta)\gamma_2}{n} + \frac{p_3'(1-\beta)}{n} \\
s \to 4, s \neq 6, 10 \quad & \frac{(1-\alpha)\gamma_2}{n} \\
6 \to 4, 10 \to 4 \quad & \frac{(1-\alpha)\gamma_2}{n} + \frac{p_3'(1-\beta)}{n} \\
s \to 5, s \neq 7 \quad & \frac{\alpha(1-\beta)\eta\epsilon\gamma_2}{n} \\
7 \to 5 \quad & \frac{\alpha(1-\beta)\eta\epsilon\gamma_2}{n} + \frac{p_1'(1-\beta)}{n} \\
s \to 6, s \neq 8 \quad & \frac{\alpha(1-\beta)\eta(1-\epsilon)\gamma_2}{n} \\
8 \to 6 \quad & \frac{\alpha(1-\beta)\eta(1-\epsilon)\gamma_2}{n} + \frac{p_1'(1-\beta)}{n} \\
5 \to 7, 6 \to 8 \quad & \frac{(K-L')\alpha(1-\beta)\gamma_2}{N-m} \\
3 \to 9, 4 \to 10 \quad & \frac{(K-1)\alpha(1-\beta)\gamma_2}{N-m} \\
7 \to 9, 8 \to 10 \quad & \frac{p_2'(1-\beta)\gamma_2}{n} \\
b \to 11 \quad & \frac{(K-1)(1-\alpha)\gamma_2}{N-m}
\end{aligned}
$$

The balance equations can be derived from the transition rates; these are not linear, but define a *fixed point* on the vector of steady-state probabilities for each region and so in practice are solved iteratively.

The probability that a processor emerging from a think state generates each action may be conveniently expressed by a table $\delta$ indexed by state and action. This is shown in Table 2.1. For example $\Pr\{\text{CR in state } 11a\} = \delta_{11a,CR} = 1 - \eta_a$, $\Pr\{\text{ DA in state } 7a\} = \delta_{7a,DA} = \alpha\eta_a(1-\beta)$ and so on.

**Message Streams**

Each action makes the processor send and receive a certain number of long or short messages through the ring. In order to calculate the time these messages take to be transmitted we need to calculate the rate at which they are produced. The next step requires us to define 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. For 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

| | *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 2.1.** State transitions from remote operations

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). Note that some messages may be destined for the local memory block of the processor and so will create no ring traffic. These are explicitly annotated with an * in the table. Thus, for the example above we have, $S_{11,AD} = 1^* + 1, S'_{11,AD} = 1^* + 1, L_{11,AD} = 1^*, C_{11,AD} = 3, C'_{11,AD} = 3$ and $M_{11,AD} = 1$. Similarly for the other actions/states. We do not list the table here, referring the interested reader to [1] for the full details.

### 2.3 Mean transmission Time

Given the above tables we can now determine the mean transmission time of a message around the ring. All messages 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 which we will assume is the same length as a short packet.

In SCI messages originating from the ring have priority over messages in the transmit queue originating from the node and so we appeal to Cobham's formulae to determine the mean transmission time of short and long messages around the ring, $T_s$ and $T_l$ respectively, on the assumption that the messages arriving at a processor are Poisson. This analysis assumes that 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, $t_{short}$ which is determined by the SCI link speed.

### 2.4 Cache/Memory Access Delay

We model the cache/memory controller as a first-come-first-served queue and appeal to a separate M/G/1 model of the queue in which the service time distribution is a probabilistic mixture of the (constant) cache and memory access times, $t_{cache}$ and $t_{mem}$ respectively. We will use the model to compare two alternative node architectures: the first is that of the conventional SCI model and the second a similar design in which the node memory is decoupled from the node cache and relocated on the main system bus, to which the processors are also attached. The latter model is thus of a collection of conventional bus-based multiprocessors interconnected using SCI and is interesting as it presents the simplest method for constructing a scalable parallel machine from existing shared-memory multiprocessors.

We first define the probabilities $P_I$ and $P_{II}$ that a memory reference addresses the cache in Regions I and II respectively:

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

Note that $P_I + P_{II} = 1 - \gamma_3\sigma$ which we denote by $P_s$ and we write $\overline{P_I} = P_I/P_s, \overline{P_{II}} = P_{II}/P_s$.

**Case 1: Standard Memory Model** The mean number of cache and memory accesses ($n_c$ and $n_m$ respectively) for each action can be found from $P_I$ and $P_{II}$, the equilibrium line state probabilities and the various tables defined earlier. From these we can find the rate at which cache ($\lambda_c$) and memory ($\lambda_m$) accesses are produced by a processor:

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

$\lambda_t = \lambda_s + \lambda_l$ is the rate at which messages are produce by a node. Short messages are produced at the rates:

$$\begin{aligned} \lambda_s &= \tau P_I \sum_{s,a} q_s \ \delta_{s,a} \ S_{s,a} \\ &+ \tau P_{II} \sum_{s',a} q'_{s'} \ \delta'_{s',a} \ S'_{s',a} \end{aligned}$$

The equation for $\lambda_l$ is identical except that $L$ and $L'$ replace $S$ and $S'$ respectively. Since the total arrival rate of cache and memory accesses is $\lambda_{cm} = \lambda_c + \lambda_m$ we can obtain the mean queuing time at the cache/memory controller from the Pollaczek-Khinchine formula:

$$Q_{cm} = \frac{\lambda_c t_{cache}^2 + \lambda_m t_{mem}^2}{2(1-\rho_{cm})}$$

where $\rho_{cm}$ is the cache/memory controller utilisation given by: $\rho_{cm} = \lambda_c t_{cache} + \lambda_m t_{mem}$ The mean time to service a memory request is then

$$\begin{aligned} T &= P_s[\frac{\lambda_s T_s + \lambda_l T_l}{\tau P_s} \\ &+ n_c(Q_{cm} + t_{cache}) + n_m(Q_{cm} + t_{mem}) \\ &+ p_{hit}\,(Q_{cm} + t_{cache})] \\ &+ \sigma\gamma_3(Q_{cm} + t_{mem}) \end{aligned}$$

where

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

The processor utilisation is then:

$$\pi = \frac{\frac{1}{\tau}}{\frac{1}{\tau} + T} = \frac{1}{1+\tau T}$$

**Case 2: Revised Memory Model** We now consider the revised memory model. 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 $\tau'$.

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:

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

The queueing time is then given by:

$$Q'_{bus} = \frac{\lambda'_{bus}\, M2_{bus}}{2(1-\rho'_{bus})}$$

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

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

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 thus:

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

## 3. Results and Validation

Some sample numerical results may be found in [1]. We are currently validating the model with respect to an execution-driven simulation of the SCI protocol running codes from the SLASH suite. SPLASH programs are executed in order to measure not only performance metrics like processor utilisation and memory access time, but also the parameters used in the model. By running the model with these parameters we can then compare the model output with that of the original executing SPLASH code. Preliminary results show a promising match for MP3D at least but this exercise is, as yet, incomplete.

## References

1. A.J. Field and P.G. Harrison. "An Analytical Model of the Standard Coherent Interface 'SCI"', Internal Report, Department of Computing, Imperial College, 1995.
2. S. Gjessing, D.B. Gustavson, J.R. Goodman, D.V. James and 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
3. The IEEE. "IEEE P1596 Standard Specification". IEEE Publication, 1989.
4. A.G. Greenberg and I.Mitrani "Analysis of Snooping Caches". *Proc. of Performance 87, 12th Int. Symp. on Computer Performance*, Brussels, December 1987.
5. 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.