.71 .31

CHAPTER 4

A METHODOLOGY FOR THE PERFORMANCE MODELLING OF DISTRIBUTED CACHE COHERENT MULTIPROCESSORS

A.J. Field and P.G. Harrison

4.1. INTRODUCTION AND BACKGROUND

In this chapter we present a general method for developing of performance models of shared-memory computer systems. These machines comprise a number of architecturally identical processing nodes whose activities are coordinated via a single global shared memory system. Each node has the ability to cache a part of the shared memory in separate high-speed stores local to each processor and the role of the coherency protocol is to ensure that each processor's view of the shared memory is the same at every instant.

In a *distributed* shared-memory system cache coherency is maintained by the interchange of messages between nodes, thereby introducing an overhead on selected read and write operations from the processor. Performance models are vital in order to understand the behaviour of speculative designs and to perform a quantitative comparison of different design options, in particular to measure the effect of the overheads of various cache coherency mechanisms which may be of interest to the designer. A change to the architecture of a processing node, for example, may affect the flow of coherency traffic and create an unexpected bottleneck. Equally, a change to the cache line states

may change the amount or nature of the memory and network traffic created for a given workload. Experiments with other hardware parameters such as cache line size are also important for optimizing static hardware parameters and cost/performance.

A commonly used modelling technique in this context is *execution-driven* simulation in which a parallel program is executed on top of a simulated memory system. These simulations can be extremely accurate but incur very long execution times and are highly prone to logical errors in coding. An execution-driven model of SCI has been developed at the University of Edinburgh[1] but this does not explicitly model the communication network.

The alternative, yet complementary, approach proposed here is that of analytical modelling using a combination of established mathematical results and numerical techniques. These analytical models are in general far more efficient computationally than the equivalent simulation. The execution time of the models we describe here are measured in seconds or minutes using Mathematica[2] on a Macintosh computer, compared with many hours, or even days, for a simulation run.

A number of such models have been developed for shared-memory systems. Modelling of bus-based multiprocessor systems has been addessed in some detail[3] and numerical predictions have been produced for a range of operating parameters and coherency protocols. A simple model of a distributed shared-memory system based on the SCI[4,5] protocol has been developed[6], but this only models the ring (the default network for SCI systems) and does not take into account the all-important coherency traffic produced by the SCI protocol itself.

We concentrate here on distributed cache coherency protocols and show how they can be modelled using a general-purpose analytical approach which can be adapted to different architectures and coherency mechanisms. The methodology focuses in particular on the coherency protocol since a specific node architecture can be modelled using standard techniques, e.g. queueing networks.

We demonstrate the approach with two case studies, the first being a model of SCI referred to above and the second a protocol, which refer to as ALITE[7], which bears many similarities also to Sun's S3MP protocol[8]. For each of these we show how the methodology is applied to yield a model, and show some sample results demonstrating the usefulness of the model for experimental purposes. We include some discussion of the queueing models for the node and network although this is not

emphasised in any detail. By way of variety, the first model includes a detailed representation of the node architecture and associated bus traffic, but a very straightforward model of the network. The second conversely employs a more substantial network model and a simplistic queueing model of the node.

The rest of the chapter is organised as follows. Section 4.2 describes the class of machines which are the subject of the work. Section 4.3 describes the modelling methodology. Sections 4.4 and 4.5 describe two case studies showing how the methodology can be applied to systems of genuine practical interest. Some numerical results are presented in Section 4.6 and the conclusions are laid out in Section 4.7.

4.2. SHARED MEMORY SYSTEMS

The simplest shared-memory systems consist of a collection of processing nodes together with a global memory all attached to a shared bus. Each processor has a local cache which stores recently accessed locations in the global address space. A *coherency protocol* ensures that the cache contents are consistent, so that each read from a processor always picks up the latest value written. This protocol exploits the broadcast nature of the shared bus which allows each cache controller to listen to all memory transactions from the other processors (*snooping*).

Bus-based machines are conceptually simple but do not scale well so much attention is now being focused on *distributed* shared memory implementations in which the memory is partitioned into a number of segments, one per processing node. The nodes are then connected by a more general, and scalable, interconnection network. The global architecture of such a system is shown in Figure 4.1. Each node now consists of a processing resource (this may be a single processor or a collection of tightly coupled processors, possibly with one or more local caches), a node cache containing copies of the remote locations that are cached at this node, a segment of the global memory, and a controller for managing network communication and remote coherency traffic. Coherency of the caches is maintained by a series of explicit point-to-point communications (c.f. snooping on a shared bus).

In order to exploit spatial locality the memories and the caches are divided into *lines* which are collections of adjacent memory locations. Lines carry with them additional *state* information which is used by the coherency protocol to determine which operations need to be

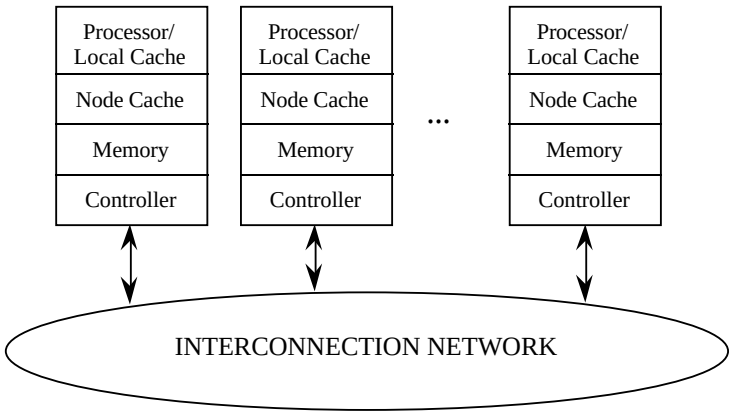

Figure 4.1: Global System Configuration

undertaken in response to each read or write to the line in order to maintain coherency. These coherency operations may change the state of one or more of the cache lines.

Data is shipped around the machine in units of lines and the protocol is usually carefully constructed so that whole lines (which may be of the order of tens or hundreds of bytes) are only transmitted between nodes when absolutely necessary. Where possible, shorter "control" messages are used.

To maintain coherency, it is neccesary to know exactly which node caches in the system contain copies of a given line in memory. This information may be contained in a table, or *directory* at the home node, or it may be distributed in the form of a *sharing list* which links (lines in) the node caches that contain copies of the cached line. Sharing lists may be either singly- or douly-linked.

4.3. MODELLING METHODOLOGY

The proposed methodology breaks down the modelling process into six steps:

1. Definition of the machine configuration and characterisation of the system workload

2. Development of a (finite state space, irreducible and positive recurrent) Markov model of the sharing list states to determine the

equilibrium probability distribution of the number of sharers of a given memory line

3. Determination of the cache line states and state transitions together with the solution of an associated Markov model of the line states to yield equilibrium line state probabilities

4. Determination of coherency operations and the associated probabilities for each in a given state for each type of memory access

5. Identification of the major traffic classes and the number of each incurred by each coherency operation in each state

6. Solution of queueing model(s), or similar, for the communication network and/or node using the traffic rates derived from the previous step

As we are focusing on the modelling of the coherency protocol, we only detail steps 1—5. There are well established techniques for modelling a variety of communication networks and node architectures so we present few specific details. Two examples are, however, briefly discussed as part of the case studies described later.

The performance measures of interest are numerous and are easily extracted from the resulting models; in what follows we focus on the probability that a processor is busy doing useful work, π and the average memory response time (or latency), R.

We now expand the details of the main steps in the methodology.

4.3.1. Preliminaries

It is assumed that each processor alternates *think periods* and periods when it waits for a memory access to be serviced. The mean think period is defined throughout to be $1/\tau_0$. After a think period, the processor generates a memory request and this request may or may not invoke a coherency operation. During the time a request is processed, the processor is idle. Thus, $\pi = \tau_0^{-1}/(\tau_0^{-1} + R) = 1/(1 + \tau_0 R)$. We will write $\tau = \pi\tau_0$ for the rate at which a processor actually generates a read/write request to the memory system.

The machine configuration parameters must be specified and in what follows we assume the following notation:

K – The number of nodes

N – The total number of shareable memory lines
n – The capacity (in lines) of each local cache

The degree of locality in the workload is characterised by its cache hit/miss rates on reads/writes, and by a parameter P_{loc} which is the probability that a memory reference from a processor is to a line whose home is in the local memory of that processor's node. The read hit and read miss rates will be written β_{rh} and β_{rm} respectively; β_{wh} and β_{wm} similarly for writes. It is also convenient to define $\beta_h = \beta_{rh} + \beta_{wh}$, $\beta_m = \beta_{rm} + \beta_{wm}$, $\beta_r = \beta_{rh} + \beta_{rm}$, $\beta_w = \beta_{wh} + \beta_{wm}$.

4.3.2. Sharers

In general, the sharers of a memory line are represented explicitly by a sharing list or similar structure. To write down the transition rates between cache line states (see below), and also to determine the average traffic generated per memory reference, we need information about the number of sharers of a given cache line. This is produced from a separate model of line sharing, taken from the point of view of the memory.

We assume that the evolution of the number of sharers of a memory line follows a Markov process, independent of the states of other memory lines. This process is irreducible, aperiodic and has a finite state space. It thus has a steady-state. The model can be solved using standard techniques since the transition rates are expressed solely in terms of known model parameters. It leads to the mean Δ, and probability distribution of the number of sharers at equilibrium.

4.3.3. Line States

The specification of a cache coherency protocol includes a definition of the possible cache line states and valid state transitions. In the first part of the modelling process we aim to determine the equilibrium probability that a cache line is in each of the defined states.

We again assume that the evolution of the state of a given line in a cache (we shall refer to this as the *observed* cache line) follows a Markov process, independent of the states of other lines.

Unlike the submodel of line sharing, however, the transition rates here may in general be defined in terms of other variables which are

themselves dependent on the equilibrium line state probabilities. For example, in a protocol which distinguishes clean data from dirty data a transition rate may depend on the probability that a given cache line (other than the observed one) is in the dirty/clean state. This probability is itself estimated from the equilibrium line state probabilities. This leads to a circularity in the definition of the transition rates and hence to a fixed point equation which is solved iteratively. We shall see examples of this circularity in both the case studies which follow.

If S is the set of line states then the equilibrium line state probabilities will be written $q_s, s \in S$.

4.3.4. Coherency Operations

The next step is to define the coherency operations for the protocol in question. These consist of the various actions required to maintain the identifiers of all the sharers of each cached line. Where explicit sharing lists are used these will include operations for creating new lists, deleting from and adding to existing ones and for reducing or updating lists, depending on whether the protocol is invalidation based or update based.

Some operations will only be valid for certain line states. For example, a reduction operation following a write hit may only take place when the line is cached in at least one other node. This information may be built into the line state, e.g. by a bit distinguishing shared/exclusive status.

In order to compute the mean traffic per memory reference we need to determine the probability that a given operation will take place in a given state. This will be zero when the operation is undefined in a particular state.

We will use Ω to denote the set of coherency operations. The probability that operation $a \in \Omega$ is performed when the observed line is in state $s \in S$ will be written $\delta_{s,a}$.

4.3.5. Node and Message Traffic

The architecture-specific models of the processing nodes and communication network require inputs representing the arrival rates of the various traffic types assumed in the model. For example, in the case of the network, there may be a number of message types used by the proto-

col, each imposing its own demand on the network resources. Similarly for the nodes, there may be a number of internal transaction types, e.g. bus, memory or cache requests, or even combinations thereof.

There are thus a number of "traffic" types of interest and the next step is to list the number of instances of each type for each state and coherency operation. This information is presented as a set of tables (or a single three dimensional table), one for each transaction type. If there are z transaction types labelled $1, 2, ..., z$ then table $Z_{s,a}^{(i)}$ will denote the number of instances of transaction type i when performing operation a in state s, $1 \leq i \leq z, s \in S, a \in \Omega$.

In order to convert this information into a set of traffic rates for a queueing model of the node or network we need also to enumerate the costs of each transaction. A second table therefore lists the time T_i associated with transaction type $i, 1 \leq i \leq z$.

Sepcifying these tables is not easy; they typically contain a lot of information at a very low level. For notational brevity, in the two case studies below the various transaction tables are coerced into a single table within which the $Z_{s,a}^{(i)}$ and T_i are implictly defined. Reconstructing the individual tables is, however, a straightforward programming task.

4.3.6. Node and Network Modelling

The final part of the process entails modelling the internal node architecture and communication network. These details are, however, decoupled from the protocol. The protocol does not extend to a definition of the processing engine(s) at each node, or to the architecture of the communication network. (Note that the SCI protocol defines a "default" network architecture, namely a ring, but this is not imposed by the standard.)

The details of the node and network submodels are therefore beyond the scope of the methodology. The techniques which can be used are, however, well established. A number of standard text books[9,10] provide some excellent examples.

4.4. CASE STUDY—ALITE

The ALITE protocol is representative of a number of protocols based on unidirectionally linked sharing lists. It is not the optimal protocol but serves as a good reference model. It is similar in many respects to

Sun's S3MP protocol[8] and the Stanford DASH protocol[11] although no direct comparisons should be made.

Although we emphasise the model of the coherency protocol, it is instructive to see how the model interacts with the queueing model of the node. This example includes a very detailed model of the node, in particular with respect to the various types of bus traffic generated by the protocol. This in turn requires the protocol model to produce traffic rates for each of these classes. We explain the latter in some detail but omit many of the details of the node model itself. The network is assumed to be contention free so that the message transmission time is proportional to the message length.

The protocol is invalidation based. On a write, this entails sending an invalidate request to the addressed cache line's home node which marks the home copy as being invalid and then forwards the request down the sharing list. With the exception of the writing node all sharing list entries will be invalidated by setting a bit in the associated line in the second-level cache. The last entry on the sharing list responds to the invalidate request by sending a completion message to the requesting node. In the event of the write being a miss this message will also carry a copy of the line's (previous) data.

When a write is complete the locally cached copy is marked as being *dirty*, i.e. inconsistent with the original home copy, and *exclusive*, meaning that it is the only valid copy. So long as the line remains in the exclusive state the processor may write to it arbitrarily without further communication.

If another processor tries to read a line that has been written by a remote processor the read request sent to the home node is forwarded to the first entry of the sharing list. This will subsequently supply a (valid) copy of the line to the reader and the sharing list links will be updated to include the new sharer at the head. At this point both copies of the line are marked as dirty and *shared*. A line can thus be read and written remotely without having to update the home copy of the line; all that is required is to maintain the sharing list. Only when the final cached copy of a line is displaced from the cache is it necessary to write the line back to memory.

If a second-level cache line forming part of a sharing list is displaced following a miss on another location which maps to the same line, the cached copy has to be "unhooked" from the list. This is achieved by sending a message to the home node which is passed down the shar-

ing list until it reaches the entry preceding the unhooking node. The identifier of this node is then passed to the unhooker which decouples itself from the list by replying with the identifier of its successor. This is used to update the list pointers in the obvious way. We assume that the pointers associated with the second-level cache lines are stored in the network controller so that pointer maintenance and traversal can be performed without generating internal bus traffic.

Note that a line may be copied into the cache at the home node. In this situation the line state is maintained as for any other cached copy except that it does not explicitly appear in the sharing list for that line. Status information at the home memory indicates whether a copy of the line is held in the local cache.

4.4.1. Line States

The second-level cache line states are therefore as follows:

1. **Home Exclusive Clean**—The line is cached at the same node as the home copy. It is the only copy and it has not been written since being cached.

2. **Home Exclusive Dirty**—As above, except that the line has been written and so is inconsistent with the home copy.

3. **Home Shared Clean**—The line is cached at the same node as the home copy and there is at least one other copy cached at another node. The cached copy is clean.

4. **Home Shared Dirty**—As above except the cached copy is dirty with respect to the home copy.

5. **Client Exclusive Clean**—The home copy of the cached line is on another node. This is the only cached copy, however, and it has not been written since being cached.

6. **Client Exclusive Dirty**—As above except that the line has been dirtied.

7. **Client Shared Clean**—Same as 3 except that the line is not at the home node.

8. **Client Shared Dirty**—Same as 4 except that the line is not at the home node.

9. **Invalid**—The line contains no usable information.

4.4.2. Coherency Operations

The coherency operations are listed below. Note that two message types are distinguished: short messages are those which contain only control information, e.g. for managing updates to a sharing list, and long messages are those which contain both control information and a line of data. Long messages are typically about an order of magnitude longer than short messages.

Creation (CR) A read or write miss on a line in state 9 not cached by other processors. A new list is created unless the read/write is initiated from the home node.

Addition (AD) A read miss on a line in state 9 but cached by other processors. The new sharer is added to the existing sharing list.

Reduction (RE) A write hit on a line in state 3—8. The sharing list of which the write is a part is reduced so that the write becomes the only element.

Deletion-Creation (DC) A read or write miss on an uncached line whose address maps to a line in state 1—8 in the cache. The reader/writer is unhooked from the list of which it is a part. A new list is created as per CR above.

Deletion-Addition (DA) A read miss on an already cached line whose address maps to a line in state 1—8 in the cache. This must first be displaced from the cache as above. The node is added to the new sharing list as in AD above.

Deletion-Reduction (DR) A write miss on an already cached line whose address maps to a line in state 1—8 in the cache. This is first displaced and the list associated with the line being written is invalidated. The reduction is similar to RE above except that last item in the sharing list returns a long message to the requester containing a copy of the line.

Invalid-Reduction (IR) A write miss on a line in state 9 cached by other processors. Reduction proceeds as per DR above.

Note that since sharing lists are singly linked, each deletion operation (DA, DC, DR) will require an unhook message to traverse half the mean length of a sharing list on average. Note also that reduction operations (DR, HR, IR) here do not require the writer to take the head of the list before broadcasting an invalidate message to all sharers.

4.4.3. Sharers

The Markov process state transition diagram for the number of sharers is shown in Figure 4.2. Note that the state 0 covers both the case where there is no cached copy of the line and the case where the only cached copy is at the home node. This state can be entered from state 1 as the result of a displacement at the (only) remote node with a copy of the line, and from any other state as the result of a write to the line from the home node. The former occurs with probability P_{miss}/n and the latter with probability $\beta_w P_{loc}/(N/K)$ where $P_{miss} = \beta_{rm} + \beta_{wm}$. The state 1 can be reached from state 0 as the result of a remote read to the line (probability $(K-1)\beta_r(1-P_{loc})/(N/K)$), from state 2 as the result of a displacement (probability $2P_{miss}/n$), and also from any other state as a result of a write to the line from any non-home node (probability $(K-1)\beta_w P_{rem}/(N(K-1)/K)$ where $P_{rem} = 1 - P_{loc}$). The transition probabilities for the general case are shown in the diagram.

The model is solved to obtain the mean length of the sharing list, Δ, and the equilibrium probability P_i that the sharing list is of length $i, 0 \leq i \leq K-1$.

4.4.4. Transition Rates

The transition rates which follow have all been divided by the factor τ which is the rate at which the processor leaves the think state. η_c and η_d are the probabilities that a line is cached in the clean/dirty states respectively and P_c and P_u are the probabilities that a line is cached/uncached respectively. These are estimated straightfowardly from the steady-state probabilites (hence the fixed point equation) and the above sharer model.

$$s \to 1, s \neq 3 \qquad \frac{\beta_{rm} P_u P_{loc}}{n}$$

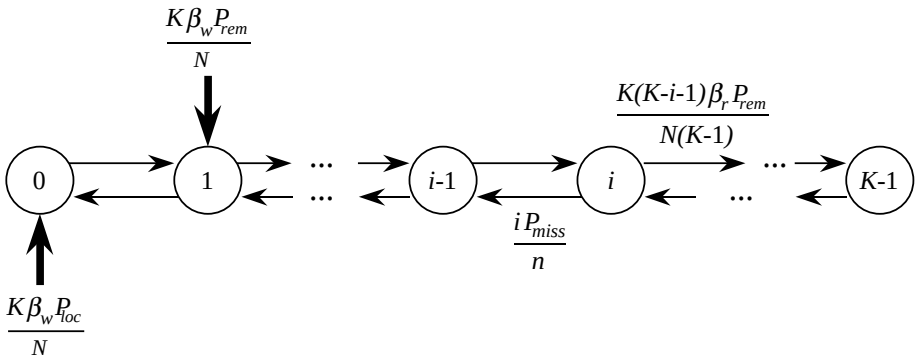

Figure 4.2: Markov model of the number of ALITE line sharers

$$
\begin{aligned}
3 \rightarrow 1 \qquad & \frac{(\beta_{rm}+\beta_{wm})P_2}{n} + \frac{\beta_{rm}P_u P_{loc}}{n} \\
1,3 \rightarrow 2 \qquad & \frac{\beta_{wh}}{n} + \frac{\beta_{wm}P_{loc}}{n} \\
4 \rightarrow 2 \qquad & \frac{(\beta_{rm}+\beta_{wm})P_2}{n} + \frac{\beta_{wh}}{n} + \frac{\beta_{wm}P_{loc}}{n} \\
5,6,7,8,9 \rightarrow 2 \qquad & \frac{\beta_{wm}P_{loc}}{n} \\
s \rightarrow 3, s \neq 1 \qquad & \frac{\beta_{rm}P_c P_{loc}}{n} \\
1 \rightarrow 3 \qquad & \frac{(K-1)\beta_r}{N} + \frac{\beta_{rm}\eta_c P_{loc}}{n} \\
2 \rightarrow 4 \qquad & \frac{(K-1)\beta_r}{N} + \frac{\beta_{rm}\eta_d P_{loc}}{n} \\
s \rightarrow 5, s \neq 7 \qquad & \frac{\beta_{rm}P_u P_{rem}}{n} \\
7 \rightarrow 5 \qquad & \frac{(\beta_{rm}+\beta_{wm})P_2}{n} + \frac{\beta_{rm}P_u P_{rem}}{n} \\
1,2,3,4,9 \rightarrow 6 \qquad & \frac{\beta_{wm}P_{rem}}{n} \\
5,7 \rightarrow 6 \qquad & \frac{\beta_{wh}}{n} + \frac{\beta_{wm}P_{rem}}{n} \\
8 \rightarrow 6 \qquad & \frac{(\beta_{rm}+\beta_{wm})P_2}{n} + \frac{\beta_{wh}}{n} + \frac{\beta_{wm}P_{rem}}{n} \\
s \rightarrow 7, s \neq 5 \qquad & \frac{\beta_{rm}\eta_c P_{rem}}{n}
\end{aligned}
$$

$$
\begin{array}{rl}
5 \to 7 & \dfrac{(K-1)\beta_r}{N} + \dfrac{\beta_{rm}\eta_c P_{rem}}{n} \\
s \to 8, s \neq 6 & \dfrac{\beta_{rm}\eta_d P_{rem}}{n} \\
6 \to 8 & \dfrac{(K-1)\beta_r}{N} + \dfrac{\beta_{rm}\eta_d P_{rem}}{n} \\
s \to 9 & \dfrac{(K-1)w}{N}
\end{array}
$$

As an example, the transition $2 \to 1$ covered by the first case occurs when there is a local read miss (β_{rm}) on a memory line which is currently uncached (P_u) and located in the local memory of the requesting processor (P_{loc}). The factor $1/n$ is the probability that the read request maps to the observed cache line. This contains a line in the state 2 before the transition, but this is displaced. Note that remote operations can also induce state transitions locally. For example, the transition $4 \to 2$ can occur if a remote processor performs a miss ($\beta_{rm} + \beta_{wm}$) on a cache line which currently holds a copy of the observed line. The transition occurs when the processor is the only other one holding a copy in the machine. We assume this occurs with probability P_2, the equilibrium probability of there being two sharers in the above Markov model. It can also happen on a write hit (β_{wh}) to the line whereupon the other sharers will be invalidated. The final term ($\frac{\beta_{wm}P_{loc}}{n}$) covers the general case transition into state 2 of a write miss on a locally held line. Finally, note that the transition $s \to 9$ corresponds to invalidation—any remote write operation to a line cached locally will cause the line to be invalidated. The factor $(K-1)$ here is the number of remote processors which can issue such a write.

4.4.5. List Operation Probabilities

The probability of each operation occurring in each state is given in Table 4.1. As prescribed, we denote the table by δ so that, for example, $\delta_{2,DC} = (\beta_{rm} + \beta_{wm})P_u$.

4.4.6. Message and Cache/Memory Traffic

To determine the various traffic types it is necessary to understand the operation of the node architecture which is shown in Figure 4.3.

	Operations						
State	CR	AD	RE	DC	DA	DR	IR
1—8	0	0	β_{wh}	$(\beta_{rm} + \beta_{wm})P_u$	$\beta_{rm}P_c$	$\beta_{wm}P_c$	0
9	P_u	$\beta_r P_c$	0	0	0	0	$\beta_w P_c$

Table 4.1: The δ Table

Here there are two buses, one of which can be servicing cache lookups from the processor whilst the other is servicing node memory references from the network. However, the processor may also need to reference memory and/or send messages to one or more remote nodes. Similarly, the network controller may need to access the cache.

There is a simultaneous resource possession problem here although, in fact, only the network controller may hold both buses at the same time to avoid deadlock. If the processor requires access to the memory bus it buffers its request, releases the cache bus and then queues separately for the memory bus. It may buffer either a short transaction (containing no copy of a line, e.g. to update the home memory status or issue a short message) or a long transaction (containing a copy of a cache line, e.g. as a result of write-back following a displacement).

This process, which is represented explicitly in the node model, requires the bus transactions to be partitioned into four Groups (i.e. service classes): those requests which require just the processor bus, those initiated by the processor which require both the cache bus and the memory bus, those which require just the memory bus and, finally, those initiated by the network which require both the memory bus and the cache bus. A Group 2 transaction which requires both buses will be split into two independent transactions, one on each bus. However the memory bus transaction is guaranteed to find the cache bus idle and so will not have to queue for it a second time after acquiring it initially. A Group 4 transaction, on the other hand, may have to contend with the processor for access to the cache bus; its service time at the memory bus is therefore augmented with a queueing time at the cache bus in the model of simultaneous resource possession used.

This particular model uses 24 transactions; 22 different types of bus

transaction, a short network message transaction and a long message network transaction. We label the classes C_1, C_2, ..., C_{23} and specify the service time for each in clock cycles. This is given in Table 4.2. Note that each class belongs to exactly one group. We write T_i to be the time (in seconds) for bus transaction C_i to complete once it has been granted the bus(es) required. These times are easily computed by multiplying the bus cycle counts in Table 4.2 by the clock cycle time, t_{clock} say. We also write T_{short} and T_{long} to denote the time taken to send short and long messages respectively.

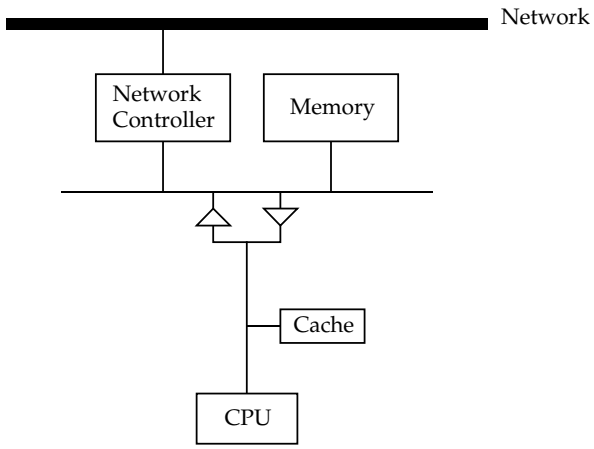

Figure 4.3: ALITE Node Organisation

The number of occurrences of each transaction type are specified in Table 4.3. The first two columns specify the operation and state. The fourth column details the bus transaction classes that are invoked in order to complete the given operation in the given state, together with the number of short and long messages sent. Since each operation/state pair may produce a number of transaction sequences depending on whether a particular line is local/remote, clean/dirty, cached/uncached etc. the corresponding probability is listed separately with each row.

Note that reductions are a special case since the reducing processor sends and receives as many short messages as there are members in the associated sharing list. We estimate the cost of this by using the mean length of a sharing list, Δ, computed from the Markov model of the sharing process.

By substituting the class descriptions from Table 4.2 in place of

Class	*Description*	*Cache bus*	*Mem bus*
Group 1			
C_1	Read from cache	26	0
C_2	Write to cache	8	0
C_3	Buffer short transaction	12	0
C_4	Buffer long transaction	32	0
Group 2			
C_5	Update memory status	3	22
C_6	Ditto and send short	3	30
C_7	Ditto and send long	3	46
C_8	Ditto and read mem to cache	27	89
C_9	Write to mem and read to cache	27	94
C_{10}	Write to mem and send short	3	85
C_{11}	Send short	3	11
C_{12}	Send long	3	27
Group 3			
C_{13}	Update home mem status	0	19
C_{14}	Ditto and send short	0	27
C_{15}	Ditto and send long	0	83
C_{16}	Write to mem and send short	0	75
Group 4			
C_{17}	Invalidate cache	10	13
C_{18}	Read from cache to network	28	31
C_{19}	Write from message to cache	29	32
C_{20}	Transfer mem to cache	26	85
C_{21}	Update mem status & complete	3	22
C_{22}	Complete operation	3	6

Table 4.2: Service classes

the C_i in Table 4.3 a descriptive breakdown of each operation/state pair is produced. For example, the bus and network traffic involved by performing operation DA (Deletion-Addition) in state 2 depends on the status and location of the new line which is to be read. There are four cases, each occurring with an associated probability.

Op.	*State*	*Prob.*	*Transaction classes invoked*
CR	9	P_{loc}	$C_3 + C_8$
		P_{rem}	$C_3 + C_{11} + C_{15} + C_{19} + S + L$
AD	9	P_{loc}	$C_3 + C_8$
		$P_{rem}P_{val}$	$C_3 + C_{11} + C_{15} + C_{19} + S + L$
		$P_{rem}P_{inv}$	$C_3 + C_{11} + C_{14} + C_{15} + C_{19}$
RE	1	1	$C_2 + C_3 + C_5$
	2	1	C_2
	3	1	$C_3 + C_6 + (\Delta - 1)C_{17} + C_{21} + (\Delta)S$
	4	1	$C_3 + C_6 + (\Delta - 1)C_{17} + C_{14} + C_{21}$
	5	1	$C_3 + C_{11} + C_{14} + C_{22} + 2S$
	6	1	C_2
	7,8	1	$C_3 + C_{11} + C_{14} + (\Delta - 1)C_{17} + C_{14} + C_{22} + (\Delta + 3)S$
DC	1,3,4	P_{loc}	$C_3 + C_8$
		P_{rem}	$C_3 + C_6 + C_{15} + C_{19} + S + L$
	2	P_{loc}	$C_4 + C_9$
		P_{rem}	$C_4 + C_{10} + C_{15} + C_{19} + S + L$
	5	P_{loc}	$C_3 + C_{11} + C_{14} + C_{20} + 2S$
		P_{rem}	$C_3 + C_{11} + C_{14} + C_{15} + C_{19} + 2S + L$
	6	P_{loc}	$C_4 + C_{12} + C_{16} + C_{20}$
		P_{rem}	$C_4 + C_{12} + C_{16} + C_{15} + C_{19} + S + 2L$
	7,8	P_{loc}	$C_3 + C_{11} + C_{14} + C_{13} + C_{20} + (\Delta/2 + 5)S$
		P_{rem}	$C_3 + C_{11} + C_{14} + C_{13} + C_{15} + C_{19} + (\Delta/2 + 5)S + L$
DA	1,3,4	$P_{loc}P_{val}$	$C_3 + C_8$
		$P_{loc}P_{inv}$	$C_3 + C_6 + C_{18} + C_{19} + S + L$
		$P_{rem}P_{val}$	$C_3 + C_6 + C_{15} + C_{19} + S + L$
		$P_{rem}P_{inv}$	$C_3 + C_6 + C_{14} + C_{18} + C_{19} + 2S + L$
	2	$P_{loc}P_{val}$	$C_4 + C_9$
		$P_{loc}P_{inv}$	$C_4 + C_{10} + C_{18} + C_{19} + S + L$
		$P_{rem}P_{val}$	$C_4 + C_{10} + C_{15} + C_{19} + S + L$
		$P_{rem}P_{inv}$	$C_4 + C_{10} + C_{14} + C_{18} + C_{19} + 2S + L$
	5	$P_{loc}P_{val}$	$C_3 + C_{11} + C_{14} + C_{20} + 2S$
		$P_{loc}P_{inv}$	$C_3 + C_{11} + C_{14} + C_{14} + C_{18} + 3S + L$
		$P_{rem}P_{val}$	$C_3 + C_{11} + C_{14} + C_{15} + C_{19} + 3S + L$
		$P_{rem}P_{inv}$	$C_3 + C_{11} + C_{14} + C_{14} + C_{18} + C_{19}$
	6	$P_{loc}P_{val}$	$C_4 + C_{12} + C_{16} + C_{20} + S + L$
		$P_{loc}P_{inv}$	$C_4 + C_{12} + C_{14} + C_{14} + C_{18} + 3S + L$
		$P_{rem}P_{val}$	$C_3 + C_{11} + C_{14} + C_{15} + C_{19} + 3S + L$
		$P_{rem}P_{inv}$	$C_3 + C_{11} + C_{14} + C_{14} + C_{18} + C_{19}$
	7,8	$P_{loc}P_{val}$	$C_3 + C_{11} + C_{14} + C_{13} + C_{20} + (\Delta/2 + 5)S$
		$P_{loc}P_{inv}$	$C_3 + C_{11} + C_{14} + C_{13} + C_{14} + C_{18} + C_{19} + (\Delta/2 + 6)S + L$
		$P_{rem}P_{val}$	$C_3 + C_{11} + C_{14} + C_{13} + C_{15} + C_{19} + (\Delta/2 + 6)S + L$
		$P_{rem}P_{inv}$	$C_3 + C_{11} + C_{14} + C_{13} + C_{14} + C_{18} + C_{19} + (\Delta/2 + 7)S + L$

Table 4.3: ALITE bus and memory transactions

Op.	*State*	*Prob.*	*Transaction classes invoked*
DR	1,3,4	$P_{loc}P_u$	$C_3 + C_8$
		$P_{loc}P_c$	$C_3 + C_6 + \Delta C_{17} + C_{20} + (\Delta + 1)S$
		$P_{rem}P_u$	$C_3 + C_6 + C_{15} + C_{19} + S + L$
		$P_{rem}P_c$	$C_3 + C_6 + C_{14} + \Delta C_{17} + C_{18} + C_{19} + (\Delta + 2)S + L$
	2	$P_{loc}P_u$	$C_4 + C_9$
		$P_{loc}P_c$	$C_4 + C_{10} + \Delta C_{17} + C_{18} + C_{19} + (\Delta + 1)S$
		$P_{rem}P_u$	$C_4 + C_{10} + C_{15} + C_{19} + S + L$
		$P_{rem}P_c$	$C_4 + C_{10} + C_{14} + \Delta C_{17} + C_{18} + C_{19} + (\Delta + 2)S + L$
	5	$P_{loc}P_u$	$C_3 + C_{11} + C_{14} + C_{20} + 2S$
		$P_{loc}P_c$	$C_3 + C_{11} + C_{14} + C_{14} + \Delta C_{17} + C_{20} + (\Delta + 3)S$
		$P_{rem}P_u$	$C_3 + C_{11} + C_{14} + C_{15} + C_{20} + 3S + L$
		$P_{rem}P_c$	$C_3 + C_{11} + C_{14} + C_{14} + \Delta C_{17} + C_{15} + C_{19} + (\Delta + 4)S + L$
	6	$P_{loc}P_u$	$C_4 + C_{12} + C_{16} + C_{20} + S + L$
		$P_{loc}P_c$	$C_4 + C_{12} + C_{16} + C_{14} + \Delta C_{17} + C_{20} + (\Delta + 2)S + L$
		$P_{rem}P_u$	$C_4 + C_{12} + C_{16} + C_{15} + C_{19} + 2S + 2L$
		$P_{rem}P_c$	$C_4 + C_{12} + C_{16} + C_{14} + \Delta C_{17} + C_{15} + C_{19} + (\Delta + 3)S + 2L$
	7,8	$P_{loc}P_u$	$C_3 + C_{11} + C_{14} + C_{13} + \Delta C_{17} + C_{20} + (\Delta/2 + 5)S$
		$P_{loc}P_c$	$C_3 + C_{11} + C_{14} + C_{13} + C_{14} + C_{18} + C_{19} + (3\Delta/2 + 6)S$
		$P_{rem}P_u$	$C_3 + C_{11} + C_{14} + C_{13} + C_{15} + C_{19} + (\Delta/2 + 6)S + L$
		$P_{rem}P_u$	$C_3 + C_{11} + C_{14} + C_{13} + C_{14} + C_{18} + C_{19} + (3\Delta/2 + 7)S + L$
IR	9	$P_{loc}P_u$	$C_3 + C_8$
		$P_{loc}P_c$	$C_3 + C_{11} + \Delta C_{17} + C_{20} + (\Delta + 1)S$
		$P_{rem}P_u$	$C_3 + C_{11} + C_{15} + C_{19} + S + L$
		$P_{rem}P_c$	$C_3 + C_{11} + C_{14} + \Delta C_{17} + C_{18} + C_{19} + (\Delta + 2)S + L$
RH	1—8	1	C_1

Table 4.3: ALITE bus and message transactions (cont.)

For example, if the line being read is on the same node as the line being displaced and is valid with respect to other sharers ($P_{loc}P_{val}$) then the operation can be completed without communication: the dirty line being displaced must be written back to memory (involving claiming both the cache and memory buses) after which the new line can be read into the cache in the opposite direction. This requires one transaction on each bus ($C_4 + C_9$) for which there is an associated total delay (see Table 4.2) of 59 cycles on the cache bus and 94 cycles on the memory bus, with 27 cycles of the latter being consumed with both buses held.

If, however, the new line's home node is elsewhere and if the home copy is invalid, due to a write operation, then more work must be done: the dirty line being displaced must be written back to memory. A long transaction is issued on the cache bus and a subsequent transaction on the memory bus writes the line to memory and sends a short message to the home node. When this is received the home node updates the pointer status field of the addressed line (adding the new node to the sharing list) and forwards a short request message to the current shar-

ing list head which has an up-to-date copy of the line. This node claims its memory *and* cache buses, fetches a copy of the line from its cache and returns it as a long message, targeted to the initiator of the DA operation. When this long message is received both buses at the initiator are claimed and the line is transferred to the cache; this in turn restarts the processor. In this latter case two short messages and one long message are sent–their transmission times do not affect the bus queueing times but do add delays to the overall read/write response time.

The (sum of the) coefficient(s) of C_i for operation p in state s, multiplied by the associated probabilities, determines the expected number of occurrences of bus transaction $i, 1 \leq i \leq 22$, on a memory access. Similarly, the numbers of short and long messages.

4.4.7. Modelling the Node

We give here a brief overview of the queueing model of the node, which has been documented in full elsewhere[12].

The node architecture is modelled as a queueing network with a server representing each bus. The bus delays depend on the type of transaction and so the transaction classes in Table 4.2 become service classes in an M/G/1 queueing model. Pointer traversal and pointer maintenance are handled by the network controller; this is pipelined and the associated delays are therefore assumed to be subsumed by the message transmission times.

The queueing network is complicated by the fact that internal requests from the processor and external requests from the network may require either one bus, or both buses, to complete a transaction. This leads to a form of simultaneous resource possession in the queueing network and hence blocking-before-service. We utilise an approximate solution to this problem by augmenting the service time at the memory bus with the waiting time at the cache bus, for the Group 4 bus transaction classes; Group 2 is similar except that there is no queueing for the cache bus, as previously described.

The response time of the memory system, R, may now be now obtained from the various bus transaction times and the short and long message transmission times t_{short} and t_{long}. Defining $\Omega = \{CR, ..., RH\}$ to be the set of coherency operations and $S = \{CXC, ..., INV\}$ to be

the set of sharing list operations, we obtain:

$$R = \sum_{p \in \Omega} \sum_{s \in S} q_s \, \delta_{s,p} \left(\left(\sum_{i=1}^{C} T_{s,p,i} \, w_i \right) + (S_{s,p} \, t_{short} + L_{s,p} \, t_{long}) \right)$$

where w_i is the total waiting time for class i bus transactions. These are computed from an M/G/1 model of the dual bus system, suitably modified to accommodate the simultaneous resource possession. The full analysis uses the generating function of the steady-state queue length distribution to determine the respective queuing times. Processor utilisation is then:

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

Note, however, that in calculating the arrival rates for each bus transaction class we assumed the existence of π by virtue of the factor $\tau = \pi \tau_0$ in the defining summation. We have thus introduced a new fixed-point problem for determining π and again appeal to an iterative solution method. We note in passing that some care has to be taken in updating the approximation to π in order to ensure convergence.

4.5. CASE STUDY—SCI

The SCI protocol[4,5] is an IEEE standard protocol based on *bidirectionally* linked sharing lists. Although there are some similarities with the ALITE protocol above, it differs significantly in the line states and in the coherency management messages required.

In contrast to the ALITE model, the model here employs a very simple model of the node but a more sophisticated model of the communication network, which is a ring (the default for SCI). It also illustrates a more sophisticated workload model which distinguishes three different classes of data. This is instructive since it demonstrates how details of this kind can be incoroprated by suitable enhancement to the line states and associated Markov process.

4.5.1. Line States

A memory line may be in two possible states

1 Home The line is not cached by any processor

2 Cached The line is cached by one or more processors

The caches hold local copies of memory lines and these copies may either be *clean* or *dirty*. The workload model here distinguishes *private* data from *shareable* data. Private data is typically stored local to the processor although this locality is formally specified by a parameter σ which denotes the probability that a private line resides in the local memory. This enables non-local private data to be modelled; such data will be cached in state 1 or 2. There are eleven basic line states:

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 line and is alone in the sharing-list.

4 Only Dirty The location contains a dirty copy of a memory line and is again 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 line.

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

8 Mid Dirty The location is in the middle of the list; 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.

The workload model assumes that there are m special cache lines which may contain shareable control variables, e.g. locks, in addition to ordinary shared data and non-local private data, whilst others may contain only the last two. The cache regions are numbered I and II respectively and are considered separately in the model. We make the assumption that the memory contains exactly m control lines in total such that there can be no displacement of one control line by another. Thus we assume 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. Note that the entries of a sharing list must all be in the same region of their corresponding caches so that within each region the cache lines are statistically identical. There is thus a separate model for each region. The parameters γ_1, γ_2 and γ_3 denote the probability that a memory reference is to a control line, an ordinary shared data line, and a prviate line respectively.

The cache line states in Region I are therefore augmented to distinguish two types of shareable data; states 3–10 are annotated with the subscript 'a' if the (shared) line contains control information and 'b' if it contains ordinary shared data. No annotations are required for Region II. Thus, for example, state 5_a indicates a clean line forming the head of a sharing list which contains a control variable. The cache lines in Region I therefore have nineteen possible states, and those of Region II eleven.

4.5.2. Coherency Operations

The coherency operations are identical to those in the ALITE architecture although there are significant differences in the way some of them are implemented. This is due in part to the fact that the sharing list here is doubly linked. There are other differences, however. For example, in order to write a line the write must, from the definition of the protocol, be at the head of the sharing list before the list can be reduced. This introduces additional traffic in some cases. As before, we label the operations {CR,...,IR}.

4.5.3. Sharers

We summarise the model for Region II since it involves fewer states and transitions than Region I. The Markov process for the number of

sharers of lines that map to Region II is shown in Figure 4.4. The structure of the model is similar to that of ALITE but lists of length i, $1 \leq i \leq K-1$, are distinguished to indicate whether or not they include the cache line at the home node. State K represents a list of length K which must include the home node and state 0 represents an uncached line. There are transitions between adjacent list states that include or exclude the home node as well as transitions between these two subsets of states caused by the home node joining or jumping off a sharing list. There are similar Markov models for lines that map to Regions I_a and I_b.

These models are solved to obtain the equilibrium probability distribution of the sharing list length, together with its mean value, $\Delta_a, \Delta_b, \Delta$.

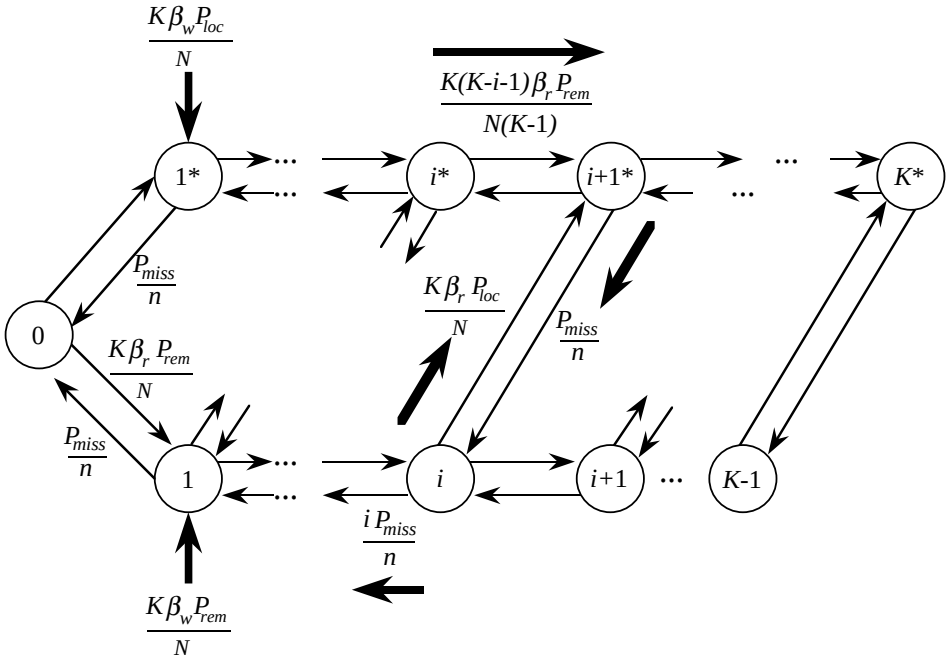

Figure 4.4: Markov model of the number of SCI line sharers

4.5.4. Transition Rates

We show below the transition rates for the cache lines in Region II. The rates for Region I can be produced similarly although there are more

of them due to the additional states within this region. The rates are defined in terms of η' which is the probability that a given line is cached somewhere, and ϵ' which is the probability that a cached line is in a clean state. These are estimated from the probability that a sharing list is empty and the line state probabilites respectively (at equilibrium), which again leads to a fixed point problem for the $q'_j, 1 \leq j \leq 11$ in Region II and $q_j, 1 \leq j \leq 19$ in Region I similarly. The rates for Region II are:

$$
\begin{aligned}
2 \rightarrow 1, s \neq 11 \quad & \frac{\alpha(1-\beta_m)\gamma_3}{n} \\
s \rightarrow 1, s \neq 1,2 \quad & \frac{\alpha\gamma_3}{n} \\
s \rightarrow 2 \quad & \frac{(1-\alpha)\gamma_3}{n} \\
s \rightarrow 3, s \neq 1,2,5,9,11 \quad & \frac{\alpha(1-\beta_m)(1-\eta)\gamma_2}{n} \\
1,2,11 \rightarrow 3 \quad & \frac{\alpha(1-\eta)\gamma_2}{n} \\
5 \rightarrow 3, 9 \rightarrow 3 \quad & \frac{\alpha(1-\beta_m)(1-\eta)\gamma_2}{n} + \frac{p'_3(1-\beta_m)}{n} \\
s \rightarrow 4, s \neq 6,10 \quad & \frac{(1-\alpha)\gamma_2}{n} \\
6 \rightarrow 4, 10 \rightarrow 4 \quad & \frac{(1-\alpha)\gamma_2}{n} + \frac{p'_3(1-\beta_m)}{n} \\
s \rightarrow 5, s \neq 1,2,7,11 \quad & \frac{\alpha(1-\beta_m)\eta\epsilon\gamma_2}{n} \\
1,2,11 \rightarrow 5 \quad & \frac{\alpha\eta\epsilon\gamma_2}{n} \\
7 \rightarrow 5 \quad & \frac{\alpha(1-\beta_m)\eta\epsilon\gamma_2}{n} + \frac{p'_1(1-\beta_m)}{n} \\
s \rightarrow 6, s \neq 1,2,8,11 \quad & \frac{\alpha(1-\beta_m)\eta(1-\epsilon)\gamma_2}{n} \\
1,2,11 \rightarrow 6 \quad & \frac{\alpha\eta(1-\epsilon)\gamma_2}{n} \\
8 \rightarrow 6 \quad & \frac{\alpha(1-\beta_m)\eta(1-\epsilon)\gamma_2}{n} + \frac{p'_1(1-\beta_m)}{n} \\
5 \rightarrow 7, 6 \rightarrow 8 \quad & \frac{(K-\Delta')\alpha\gamma_2}{N-m}
\end{aligned}
$$

$$3 \to 9, 4 \to 10 \qquad \frac{(K-1)\alpha\gamma_2}{N-m}$$

$$7 \to 9, 8 \to 10 \qquad \frac{p_2'(1-\beta_m)\gamma_2}{n}$$

$$s \to 11, s \neq 1, 2 \qquad \frac{(K-1)(1-\alpha)\gamma_2}{N-m}$$

Δ' is the mean length of a sharing list in Region II.

4.5.5. List Operation Probabiities

The probability of each coherency operation occurring in each state in Region II is given in Table 4.4. We write this δ', remarking that a similar table δ exists for Region I. Note that the notation $\overline{x}$ denotes $1-x$.

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

Table 4.4: The δ' Table

4.5.6. Message and Cache/Memory Traffic

The model here uses a straightforward queueing model of each node, but incorporates a more sophisticated model of the network than the previous case study. Node traffic is divided into two classes for simplicity, namely traffic directed toward a node cache and traffic directed toward memory. Essentially the bus transaction classes which were detailed individually in the ALITE study have been aggregated in this model.

The network has a unidirectional ring architecture. We again identify two classes of message (short and long messages as above) but here use their predicted frequencies to generate inputs to an M/G/1 model of the ring. Table 4.5 shows the total cache, memory and message traffic generated at a node by each operation in each state. Since there are only two transaction classes for the node we list them explicitly in the table (c.f. Table 4.3 referred to above) which 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.

Op.	*State*	*Short*		*Long*	*Cache*		*Mem*
		I ($r = a, b$)	II		I ($r = a, b$)	II	
		(S)	(S')	(L)	(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 4.5: SCI cache, memory and message transactions

Note that some messages may by-pass the ring, specifically if they are directed toward the same node as the initiator. These are labelled with a '*'. Shareable data is assumed to be uniformly distributed across the nodes of the machine. 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. Thus, $\Delta'_M \leq K-1$ whereas $\Delta'_H \leq K$. 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. The various Δ values are derived from the Markov models of the sharing processes in each Region/area.

4.5.7. Modelling the Node

The ring model assumes that a short message issued by a transmitting node will perform one full circuit of the ring (i.e. through $K-1$ ring buffers). For SCI in practice, the destination forwards an *echo* packet on receipt of a message. However, these are of similar length to short messages, so we model it as though the short message makes a full circuit of the ring. A long message is converted to a short echo message once it has reached the receiving node.

Messages originating from the ring have priority over messages in the transmit queue originating from the node so the model uses Cobham's formulae to determine the mean waiting times of messages in the transmit queue and ring buffer. Table 4.5 is first used to determine the rate at which messages are generated by the transmit queue and ring buffer of each node:

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

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

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:

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

Since the length of a long message is a multiple M of the length of a short message the various transmission times can be expressed in terms of the transmission time of a short message, denoted by t_{short}.

The M/G/1 model uses the first and second moments of the service time in order to compute the mean queueing time at the ring buffer and transmit queue, Q_r and Q_t respectively. This straightforward analysis finally yields the mean transmission time of short and long messages, respectively:

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

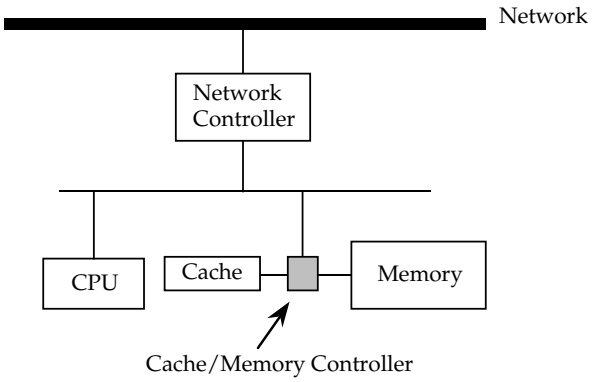

Figure 4.5: SCI Node Architecture

The cache/memory controller in Figure 4.5 is represented by a single shared queue. The requests are generated both by the processor(s) and the SCI ring via the network controller 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 can 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, as given above. The number of cache and memory accesses per operation/state are given in the tables C, C' and M defined earlier and from these we can calculate the rate at which cache and memory accesses are produced by a processor (λ_c and λ_m repectively):

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

The total arrival rate of cache and memory accesses is $\lambda_{cm} = \lambda_c + \lambda_m$ which is used to compute the mean queuing time at the cache/memory controller, Q_{cm} using the Pollaczek-Khinchine formula.

We finally obtain the mean time to service a memory request:

$$\begin{aligned}R &= (1-\sigma\gamma_3)[\frac{\lambda_s T_s + \lambda_l T_l}{\tau(1-\sigma\gamma_3)} + n_c(Q_{cm}+t_{cache}) + n_m(Q_{cm}+t_{mem}) \\ &+ p_{hit}\ (Q_{cm}+t_{cache})] + \sigma\gamma_3(Q_{cm}+t_{mem})\end{aligned}$$

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:

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

The processor utilisation is obtained from the same formula as was used in the ALITE case study.

4.6. SOME NUMERICAL RESULTS

The two models described were implemented in Mathematica 2.2 and executed on a Power Macintosh 7100/66. Each computed the equilibrium probability distributions for the line state probabilities and the number of sharers of memory lines, and hence node utilisation, system throughput and average memory latency. Numerous experiments are possible and many have been carried out to predict, for example, the quantitative effects on performance of varying cache hit rates and locality of reference. The latter includes the proportion of memory accesses to the system variables that control data structures in the SCI

model, represented by the parameter γ_1. Here we consider the processor utilisation and average memory latency for the ALITE and SCI models with parameters chosen based on their specifications together with typical, observed workload characteristics. As a baseline parameterisation, we assume uniform memory access, i.e. a memory access addresses each node with equal probability. Although this is not realistic, it provides a benchmark against which to compare alternative architectures and also generates more sharing with which to test the model than a locality-tuned, real application.

4.6.1. ALITE

Graphs of node utilisation and memory latency against the number of nodes (up to 32) were plotted for various memory sizes; 1, 10, 100 and 1000 times the node cache size (Figures 4.6 and 4.7). These suggest that the architecture scales well, especially with large memory, in that utilisation stays high and even memory latency only increases by a few percent, except with relatively small memory size. The implication is that larger memory results in fewer invalidations and hence cleaner lines and less sharing list maintenance. This is a somewhat optimistic interpretation in that the hit ratio was *fixed* at 0.9. Hence the decrease in the number of invalidations was not counterbalanced by an increased miss rate which might be expected from scaling up an application. Nevertheless, a not inconsiderable benefit in exploiting locality (giving a high hit ratio) is indicated.

4.6.2. SCI

The performance of the SCI protocol (Figures 4.8 and 4.9) degrades considerably as the number of nodes increases, utilisation falling below 0.2 in a ring of 32 processors at small memory sizes. Whilst not an entirely realistic scenario, this result indicates the serious overhead that can result when sharing lists grow. It is the contention in ring buffers that prevents a ring interconnect from scaling linearly, and here we see the effect of significantly greater than zero mean queue lengths—around 4 for a 16-node ring and over 5 for 32 nodes. Notice that sharing list lengths do not increase dramatically (actually close to logarithmically) with the number of nodes. This is because of a significant write probability (0.2) which periodically resets the length of a list to one.

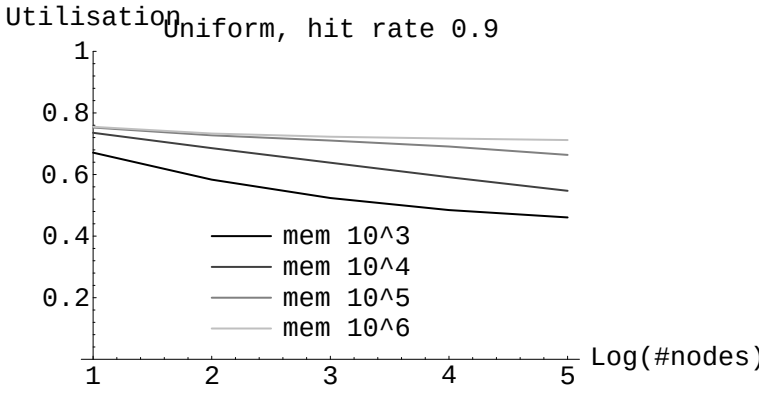

Figure 4.6: Processor utilisation curves for the ALITE protocol

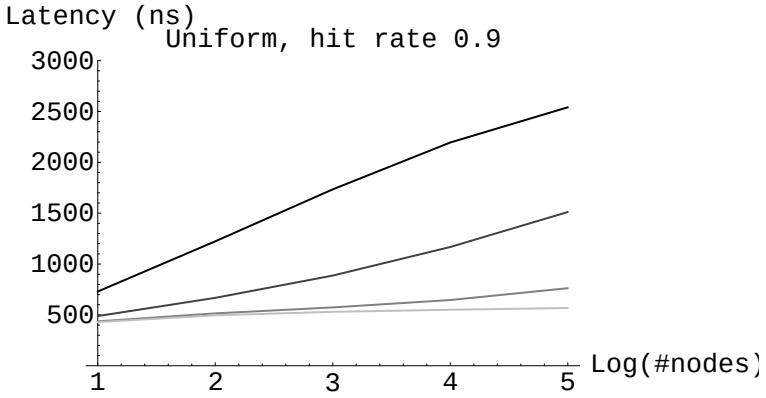

Figure 4.7: Memory latency curves for the ALITE protocol

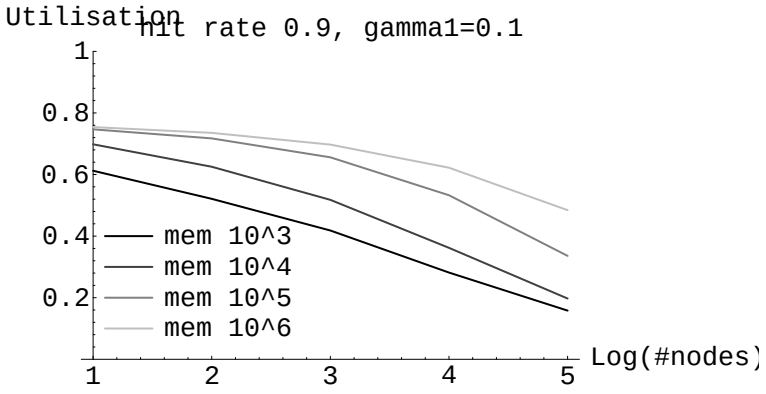

Figure 4.8: Utilisation curves for the SCI protocol

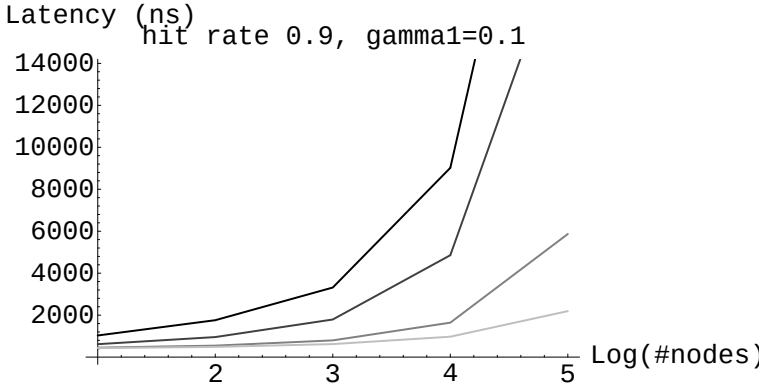

Figure 4.9: Memory latency for the SCI protocol

4.6.3. Comparison of the protocols

Since the SCI protocol involves communication over multiple ring links, the performance of the protocols themselves was compared by setting the message transmission times to zero. Network contention therefore ceases to be an issue. The predicted performance of the SCI system remains inferior to that of the realistically parameterised ALITE model as is evidenced by Figures 4.10 and 4.11 for SCI and Figures 4.6 and 4.7 for ALITE. This is due to the additional cache and memory accesses involved with managing doubly linked sharing lists, and also with the additional overhead of taking the head of the list prior to writing. Although the ALITE sharing lists require unhooking to be mediated through the home node, in practice the sharing lists are small so that, in fact, the ALITE protocol overheads are lower than those of SCI.

When message transmission times are set to zero, the ALITE model scales better, although the curves for the smallest memory size still degrade significantly as the number of nodes increases. In fact, they are not so different from those of Figures 4.6 and 4.7. This is due mainly to the dominance of the bus-contention arising from increasing invalidation traffic.

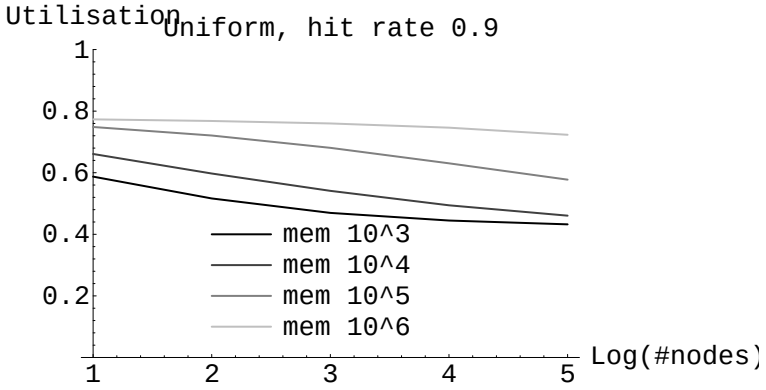

Figure 4.10: Utilisation curves for SCI with $t_{short} = t_{long} = 0$

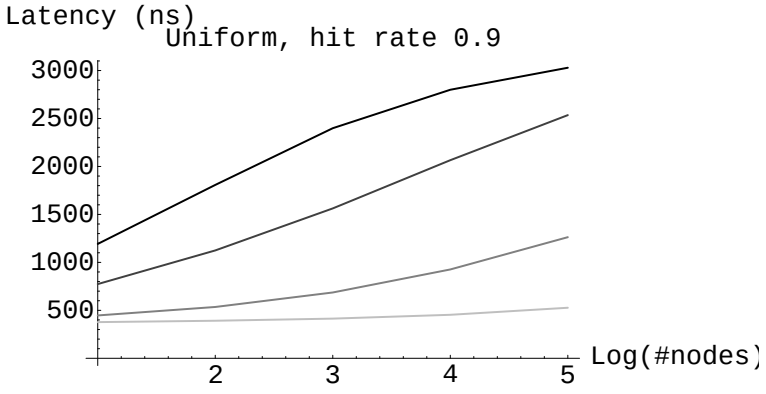

Figure 4.11: Memory latency curves for the SCI protocol with $t_{short} = t_{long} = 0$

4.7. SUMMARY AND CONCLUSIONS

We have described a general methodology for modelling the performance of distributed shared-memory multiprocessors, focusing in particular on the modelling of the cache memory system. The methodology breaks down into six steps, the first five of which are specific to the coherency protocol and the last of which relates to the modelling of the procesing nodes and communication network.

We have found the technique to be applicable to a range of distributed shared memory systems and have illustrated its use here with two case studies. The two examples use different coherency protocols and have different node and network architectures. Between them they demonstrate the flexibility of the method, in particular in coping with radically different node and network models. An arbitrary level of detail relating to the memory, cache, node bus and network traffic can be captured in the form of transaction class tables which collectively detail the amount of "work" required to perform each coherency operation in each state.

One of the benefits of a stochastic model in this context is the relative ease with which the effect on performance of various design changes, or alterations to the assumed workload, can be analysed. For straightfoward changes, this is often simply a case of changing the model pa-

rameterisation. However, more drastic alterations to a coherency protocol, for example, or the way it is implemented in hardware, are often easily remodelled by small changes to the line states and/or the state transition rates, or by a reworking of the transaction tables. We have presented few details of experimental work with the models described here since this is not the essence of the work, but remark that in our experience the approach we have described helps significantly to identify where and how model changes need to be made. We have found, for example, that certain protocol optimisations can be modelled by small methodogolical changes to the line states and associated transitions, the "δ" table and transaction tables. One change neatly leads to the next and the whole process can take as little as a few minutes from start to finish. The benefits to a designer of the type of system in question are self evident.

References

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

This paper summarises some of the work described in the first author's PhD thesis, including the development of an execution-driven simulation model of an SCI-based multiprocessor. The simulator code does not, however, model the SCI ring explicitly.

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

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

This is a classic paper on coherency protocol modelling and one which inspired much of the work in this paper. It develops models for a range of bus-based protocols based on Markov models for the line states. A comparitive analysis of the various protocols is undertaken and the results are validated against a simulation model.

[4] 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

This paper presents an overview of the SCI protocol. It is essentially a condensed summary of the full IEEE specification document cited below, but greatly abstracted to explain the key principles whilst avoiding low-level implementation details.

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

This is the full IEEE specification for SCI. The specification is accompanied by a C program code which collectively constitutes a chip-level simulation of an SCI node. The specification code is very low level, however, and this makes it unsuitable as the basis for a practical SCI simulation model.

[6] 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.

This paper is one of the first papers to address analytical modelling study of SCI. It presents a detailed model of the SCI ring, which is represented by an M/G/1 queue. The paper does not model coherency traffic, however.

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

This paper is principally concerned with a new proposal for a coherent cache-only memory architecture (COMA), but includes in its discussions a reference model of a CC-NUMA protocol, ALITE, which has been referred to in this paper. The main idea behind the COMA architecture proposed is to use existing TLB harwdare for virtual memory management to perform page-level associative memory look-up.

[8] A. Nowatzyk *et al*, "The S3.mp Scalable Shared Memory Multiprocessor", *in Proc. of the 1995 International Conference on Parallel Processing*", Oconomowoc, Wisconsin, August 1995, pp. I1–I10.

This paper won the best paper award at the 1995 International Conference on Parallel Processing. It summarises the architecture of Sun's proposals for workstation networks supporting distributed shared memory using additional controller cards and an interface to an optical communication network. The coherency protocol supported has some similarities with the ALITE protocol referred to in this paper.

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

This is a detailed and comprehensive text book covering a wide range of state-of-the-art techniques in analytical performance modelling.

[10] E.D. Lazowska, J. Zahorjan, G.S. Graham and K. Sevcik, "Quantitative System Performance". Prentice Hall. 1984.

This is a more practically oriented, yet very comprehensive, exposition of the state-of-the-art at the date of publication.

[11] M. Thapar and B. Delagi, "Stanford Distributed-Directory Protocol", IEEE Computer, Vol.23, No. 6, June 1990, pp. 78-80.

This paper describes Stanford's own coherency protocol, which is similar in many ways to the ALITE protocol described here. It serves as an excellent reference model as well as being a functional protocol in its own right.

[12] A.J. Bennett, A.J. Field and P.G. Harrison, "Modelling and Validation of Shared Memory Coherency Protocols", Internal Report, Department of Computing, Imperial College, 1996.

This provides a detailed description of the model of the ALITE coherency protocol referred to. Also included is a preliminary model validation against an execution-driven simulation model of the ALITE memory system.