Analytic Modeling of Cache Coherence Based Parallel Computers

Kazuki JOE and Akira FUKUDA

Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

In this paper, we propose an analytic model using a semi-markov process for parallel computers which support cache coherence mechanism by hardware. The proposed model, the Semi-markov Memory and Cache coherence Interference Model, can be used for the performance prediction of cache coherence based parallel computers since it can be easily applied to descriptions of the waiting states for network contentions or memory interferences of both normal data accesses and cache coherence requests. Conventional analytic models by stochastic process for parallel computers have an unavoidable disadvantage of the explosions of the number of states as the system size is enlarged even if they are for simple parallel computers without cache memory. The number of states required by constructing our proposing analytic model, however, does not depend on the system size but only on the kind of cache coherence protocol. For example, the number of states for the synapse cache coherence protocol is only 19 as is described in this paper.

共有メモリ型並列計算機の解析モデル

城 和貴 福田 晃

奈良先端科学技術大学院大学情報科学研究科
奈良県生駒市高山町８９１６－５

Abstract

キャッシュ・メモリを有した共有メモリ型並列計算機に対する、セミマルコフ過程を利用した解析モデルを提案する。提案される SMCI (Semi-markov Memory and Cache coherence Interference) モデルは、キャッシュ・コヒーレンス制御命令や通常のデータ・リクエストの、ネットワーク競合やメモリ競合による待ち状態を、実際の並列プログラムに特有のパラメータを与えることにより、容易に記述できる。また、並列計算機自体のシステム構成も入力パラメータとして与えることができるため、さまざまなシステム構成を持つ並列計算機の、アプリケーションごとの性能評価を容易かつ詳細に得ることができる。これまでのマルコフ連鎖等を利用した並列計算機の解析モデルと異なり、SMCI モデルにおける状態数は、プロセッサ台数やメモリ・サービス時間に依存せず、評価対象とする並列計算機の採用しているキャッシュ・コヒーレンス・プロトコルによってのみ決定する。従って、SMCI モデルの計算時間は本質的にシステムの規模に左右されない。

1 Introduction

Parallel computers are becoming larger and more complicated through results of research into such things as cache coherence protocols [1], processor clustering schemes [6], scalable shared memory mechanisms [7, 10], etc.

To evaluate the performance of such large and complicated parallel computer systems, one of the possible methodologies is simulation. However, simulating a large, e.g. thousands of processors, parallel computer system leads to another problem of the computational cost. For example, we can not execute any simulation program which simulates a parallel computer with 1,024 processors on a typical workstation.

Analytic modeling is widely known as a valid and inexpensive methodology for the evaluation of parallel computer systems. The problem is that most of the previous works on analytic modeling, which were mostly by using of a markov chain [2, 8, 11] or queueing theory [9], do not support parallel computers with cache memory.

Early work related to this research was done by Bhandarkar [2] of which markov chain based model was applied to a parallel computer to study the memory interference. The state definitions of his model were represented by tuples of the number of processors which were issuing memory requests to corresponding memory modules. The problem of his model laid in the explosions of the number of states as the system size is enlarged.

Marsan proposed a simplification of the similar kind of analytic models by converting the exact markov chain into a simple birth-death process. The number of the states was reduced to the same number of processors [8]. Furthermore, Mudge proposed more simplified approach to the similar kind of model by using of a semi-markov process. Finally the number of the states reached to only 4 [12]. These methodologies, however, have not been applied to analytic modeling of cache coherence based parallel computers because they are too complicated to keep the number of their states small.

Using stochastic models, some work focused on the cache coherence mechanism [15] but did not support the network contention nor the memory interference that took place in parallel computers. Besides stochastic models, queueing network models with the MVA method have been investigated for the performance prediction of parallel computers with cache memory [14]. Although the MVA method based analytic model is another recent trend [5, 3], we have interests in exploring the stochastic model based methodology.

In this paper, the Semi-markov Memory and Cache coherence Interference (SMCI) model is proposed, which can analyze the cache coherence mechanism as well as for the network contention and the memory interference both for data and cache management requests of cache coherence based parallel computers. The advantage of our proposing model is its small number of the sates. When the SMCI model is applied to a parallel computer with thousand of processors which supports cache coherence mechanism by the synapse protocol [1], the number of states is only 19.

The structure of this paper is as follows. The assumptions for the SMCI model of the whole system, the definition of various processor requests and the caching probability of a shared block are described in Sect.2. Actual states and internal variables are defined and the method for calculating the model is shown in Sect.3.

2 Preliminary

2.1 The Policy of Modeling

Previous work for modeling a parallel computer without cache memory was done by investigating a processor's state transition, assuming that there were multiple identical state transitions, and constructing the whole system model. However it is difficult to construct a model of a parallel computer with cache memory allowing for the cache coherence mechanism because the state transition of the whole system can be affected by some behavior of a single processor. When a processor makes, for example, a write request during a system cycle, the behavior of the whole system may be determined by the state of the requested data block. If no other cache holds a copy of the requested data block, the request is typically a *write miss*. If some cache holds a copy of it as *clean*, the request causes the cache to issue an *invalidation*. If a copy of the requested data block is in the local cache and already has been modified, the request is simply a *write hit*. Thus to model the whole system, including such events, it is necessary to predict each state of a shared block in each system cycle, some caches hold a copy of the block or not, whether the held copy is *clean* or not, etc.

In this paper we start from the cache miss rate of shared blocks derived from Wang's model [15]. Using the miss rate, we approximate the prediction of the states of a shared block and define the state of a processor under this prediction with a semi-markov process (**SMP**. See [13]). Then, we formulate the relation between the limiting probabilities of the state of a processor and the processor request rate. Given an initial value to the processor request rate, the limiting probabilities can be calculated. Based on them, the processor request rate is updated. Repeating these calculations, we can find a convergent point for the request rate which indicates the steady states of the system. The effectiveness of the SMCI model is demonstrated by comparing its results with the simulation results of [1]. The results show that there are only 8.0% differences between the actual simulation and our analytic model while our analytic model can predict the performance of a 1,024 processor system in the order of micro second. We do not describe these results in this paper because of a lack of space.

2.2 The Assumption of the Model

The target parallel computer consists of several processors, their own private cache and memory modules connected by a single shared bus network. The cache coherence mechanism with the Synapse protocol [1] is supported by hardware. The reason of adopting this protocol is its small number of states. Also the bus network is required to compare with the simulation results. We would like to remark that the SMCI model can be easily applied to other networks including multi-stage networks. See our another work [4].

To simplify the analytic model, the following assumptions are introduced:

1. The behavior of the processors can be modeled as identical stochastic processes.

[1] Although Synapse protocol uses *INVALID*, *VALID*, *DIRTY* as the state of blocks, we will use *Invalid*, *Clean*, *Dirty* for the rest of this paper. The reason is historical one. Actually *Clean* is the same to *VALID*.

2. The duration of the data request (except cache coherence control requests) is an independent, geometrically distributed discrete random variable with mean λ. And the request service times are given determinately by a architecture specification.

3. The running program is in a steady state; the initial paging for loading or the initial poor cache hit rate are not considered. But the replacement of a cache block when a cache entry is full is considered.

Assumptions (1) and (2) are necessary to construct the analytic model. Assumption (3) is derived from our use of the limiting probability in Sect.2.3

2.3 A Semi-Markov Process

In this paper an SMP is a discrete stochastic process which consists of K states. In state i, it sojourns for the period given by the time distribution function $F_{ij}(t)$ and makes a transition to state j with probability p_{ij}. The average sojourn time η_i is expressed as below.

$$\eta_i = \sum_{j=1}^{K} p_{ij} \sum_{n=0}^{\infty} t_n F_{ij}(t_n) \tag{1}$$

If an SMP has an ergodic irreducible embedded markov chain (**EMC**), the limiting probability of state i can be formulated as below.

$$P_i = \frac{\pi_i \eta_i}{\sum_{j=1}^{K} \pi_j \eta_j} \tag{2}$$

where $\{\pi_i\}$ is the limiting probability of an EMC. The model described in this paper consists of one ergodic EMC, so Eq.(2) is available.

2.4 The Definitions of the Various Request Rates

In a parallel computer the request rate from a processor is determined when it runs. It should consist of the original request rate from the given program and the reissued request rate from processor waiting states because of network contentions. To construct an analytic model for a cache coherence based parallel computer, we give new definitions for four various request rates: **Normal request rate** φ_{normal} is the rate that a processor issues a request just after its computation state. **Memory request rate** φ_{memory} is the rate that a processor issues a request to some data in actual memory and is defined as (normal request rate) $\times$ (cache miss rate) + (the same request rate from its waiting state). **Coherence request rate** $\varphi_{coherence}$ is a sum of the request rate for cache coherence control and the same request rate from its waiting state. **Network request rate** $\varphi_{network}$ is the rate that a processor issues to the network from any states and is defined as (memory request rate) + (coherence request rate).

In this paper each request rate is a rate of requests in a network cycle time.

2.5 The Caching Probability of a Shared Block

To know whether a request from a processor causes the processor to issue cache coherence control requests, we need to predict if the requested shared block is cached by another cache. If the number of cached shared blocks in a cache is L, and the total number of shared blocks is S, the prediction can be defined as the probability of L/S.

We use S as an input parameter for the SMCI model. Before obtaining L, we must determine the cache hit rate, H, for shared blocks. As described in Sect.2.1, we use the result (bellow equation) of [15] as an approximation where the cache miss rate for Synapse protocol is given.

$$H = 1 - \frac{1}{ls} \frac{P(P-1)(1-r)}{(P-r)(1+(P-1)(1-r))} \tag{3}$$

where P is the number of processors, r is the read request rate, and ls is the access burst length [15]. The access burst length depends on program model.

We can not obtain the number of cached shared blocks in a cache by simply multiplying the total number of shared blocks and H since the access pattern for shared blocks is not always uniformly distributed. In this paper we use Archibald's assumption [1] as the access pattern. The probability that a shared block is in ith entry of the LRU stack is $(1/(5+i) - 1/(6+i))g$ [1] where g is a normalizing coefficient. Therefore we can obtain L by the following equation.

$$\frac{\int_0^L g(\frac{1}{5+x} - \frac{1}{6+x})dx}{\int_0^S g(\frac{1}{5+x} - \frac{1}{6+x})dx} = H \tag{4}$$

After some simplification, we obtain L as follows.

$$L = \frac{6\frac{5}{6}\left(\frac{6(5+S)}{5(6+S)}\right)^H - 5}{1 - \frac{5}{6}\left(\frac{6(5+S)}{5(6+S)}\right)^H} \tag{5}$$

3 The SMCI Model

3.1 The Definition of the States

To construct the SMCI model, we first define each state for an SMP. Table 1 shows definitions of the actions of a processor to its cache, memory and network. This state definition is based on the Synapse protocol.

3.2 The State Transitions

Figure 1 shows the transitions between the defined states. A state transition occurs when any kinds of request (including requests due to cache memory accesses) is issued. In this figure, h represents the cache hit rate for private blocks, H represents the cache hit rate for shared blocks, r represents the read request rate to read/write requests, d represents the probability that the requested block is *dirty*, c represents the probability that the requested block is not *dirty* , w represents the probability that the request is not accepted, u represents the access rate for shared blocks to all blocks, x represents the probability that the cache has already been exhausted,

表 1: State Definitions in the SMCI Model

COM	**COM**putation
Rh	Reading from cache (**R**ead **h**it)
Wh	Writing to cache (**W**rite **h**it)
$WhIV$	Invalidation caused by Wh (**Wh**-**I**n**V**alidate)
$\overline{WhIV}$	The waiting state for $WhIV$
Rc	Cache miss read from non *dirty* block (**R**ead **c**lean)
$\overline{Rc}$	The waiting state for Rc
Rd	Cache miss read from *dirty* block (**R**ead **d**irty)
$\overline{Rd}$	The waiting state for Rd
Wc	Cache miss write to non *dirty* block (**W**rite **c**lean)
$\overline{Wc}$	The waiting state for Wc
$WcIV$	Invalidation cause by Wc (**Wc**- **I**n**V**alidate)
$\overline{WcIV}$	The waiting state for $WcIV$
Wd	Cache miss write to *dirty* block (**W**rite **d**irty)
$\overline{Wd}$	The waiting state for Wd
WB	Writeback caused by an invalidation (**W**rite**B**ack)
$\overline{WB}$	The waiting state for WB
RP	Writeback caused by block replacement (**RP**lace)
$\overline{RP}$	The waiting state for RP

m represents the probability that the replacing block has been modified, and y represents the rate of data requests to all requests (including writeback requests) from the COM state.

COM state changes only when requests occur. It changes to WB, $\overline{WB}$, Rh, Rc, Rd, $\overline{Rc}$, $\overline{Rd}$, Wh, Wc, Wd, $\overline{Wc}$ and $\overline{Wd}$, depending on whether the requests are for writeback caused by invalidations, whether the request is read or write, whether a cache hit occurs, whether the addressed block is *dirty*, or whether the network is busy respectively.

Rh, $WhIV$, $\overline{RP}$ and $\overline{WB}$ change only to COM. Wh changes to $WhIV$ or $\overline{WhIV}$ depending on the traffic when the addressed block is not *dirty* . When the block is *dirty*, it changes to COM.

Rc changes to COM when there are *invalid* blocks in the cache. Even if there is no *invalid* block, when the replace candidate is not *dirty*, the processor does not use network. So Rc can change to COM directly. When there is no *invalid* block and the replace candidate is *dirty*, Rc changes to RP or $\overline{RP}$ depending on the network traffic to writeback the replaced block. Rd, Wd and $WcIV$ changes the same as from Rc. Wc changes to $WcIV$ or $\overline{WcIV}$ depending on the network traffic to issue the invalidation request.

The waiting states, $\overline{WhIV}$, $\overline{Rc}$, $\overline{Rd}$, $\overline{Wc}$, $\overline{WcIV}$, $\overline{Wd}$, $\overline{RP}$ and $\overline{WB}$ changes to $WhIV$, Rc, Rd, Wc, $WcIV$, Wd, RP and WB respectively or changes to themselves depending on network traffic.

In Fig.1, $a1 = y(h(1-u)+Hu)r$ gives the probability that COM changes to Rh. The derivation is as followings: When a request occurs in COM, it must be a data request (with rate y) or a writeback request (with rate $1-y$). If it is a data request, it must be a read request (with probability r) or a write request (with probability $1-r$). If it is read one, the request must be satisfied at the cache. When the request is bound for private blocks (with probability $1-u$), the private block cache hit rate is h. Otherwise it must be bound for shared blocks (with probability u) and the shared block cache hit rate is H. Thus we obtain $a1 = y(h(1-u)+Hu)r$. We do not explain other elements of the state transition probability because of lack of space.

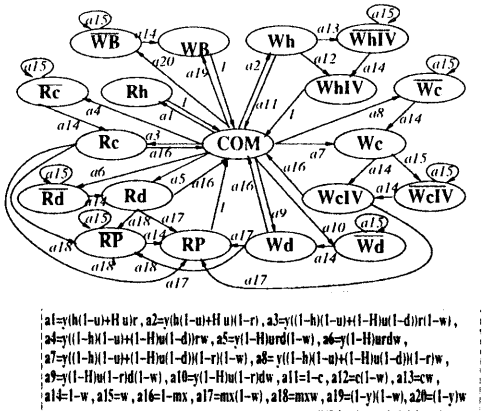

図 1: The state transition graph of an SMP based on Synapse protocol.

3.3 The Construction of an SMP

Before constructing an SMP, the distributions of sojourn time for each state of a processor are required. The states of a processor can be classified into three groups with regard to the distributions of their sojourn time: 1) the computation state, 2) the request service states for cache or memory, and 3) the waiting states. As described in Sect.2.2, we assume that the sojourn time for computation is geometrically distributed with parameter λ. Also the sojourn time for the request service states is determinately given by the below function $F_{ij}()$.

$$F_{ij}(t) = \begin{cases} 1 & \text{if } t \geq \text{The access time of state } i \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

For example, the access time for a cache read, memory read and invalidation is 1, 16 and 4 respectively. These values are from specifications of target architectures. Furthermore, we assume that the sojourn time for the waiting states is geometrically distributed with parameter Wt where the average waiting time, Wt, is given latter.

The stochastic process $X = \{X_n\}_{n=1}^{\infty}$ defined by the above state space and transition probabilities represents the behavior of a processor. It is trivial that X is a markov chain because of assumptions in Sect.2.2. The

sojourn time of each state of X is given by the above time distributions. Considering each sojourn time as a random variable sequeces $T = \{T_n\}_{n=1}^{\infty}$, the stochastic process $(X, T) = \{X_n, T_n\}_{n=1}^{\infty}$ constructs an SMP.

It is obvious from Fig.1 that the EMC X of the SMP (X, T) is ergodic. Thus we can use Eq.(2) to get the limiting probability. As described in Sect.2.1, we start from an appropriate initial request rate, obtain the limiting probabilities, update the request rate, obtain the limiting probabilities and repeat these calculations until the request rate is saturated. Thus we oftain the steady states of the system.

3.4 The Derivations of Variables used in the SMCI Model

To evaluate the SMCI model, some appropriate input parameters are needed. They are h, r, u, P, S and m, which are the same ones as in Archibald's simulation. Also each sojourn time for the computation and request service states is given as input parameters.

The aim of this subsection is to define the internal variables by input parameters. The internal variables are c, d, x, w, φ_i (several reuqest rates) and sojourn time for the waiting state Wt. Notice that H and L are given by Eq.(3) and (5) respectively.

The probability, c, indicates whether an invalidation request is issued when a write hit occurs. The invalidation should be issued when the requested block is a shard block with the clean state or when it is not a shared block which has not been modified yet. Thus c is given below.

$$c = (1-u)umd + u\left(1 - \left(1 - \frac{Lr}{S}\right)^{P-1}\right) \quad (7)$$

Since L/S represents the probability that a shared block is in a cache and it was caused by a read request, $(1 - L/S)^{P-1}$ represents the probability that there are no shared blocks, which was caused by read requests, in other $P-1$ caches. umd is the probability that a private block in a cache has not been modified yet, and is given by $umd = 1 - (1-h)(m+r-1)/((1-r)h)$, of which derivation is from [1].

The probability, d, that another cache contains the *dirty* shared block is:

$$d = (P-1)\frac{L(1-r)}{S}\left(1 - \frac{L(1-r)}{S}\right)^{P-2} \quad (8)$$

The probability, x, that there are no cache entries when a cache miss occurs is obtained by the following procedure. x is also the probability that there is no invalidation request when exhausting cache entries because only invalidation makes a cache entry become empty. Here, we assume that the empty cache entry exists in a cycle only if an invalidation request was issued to the cache at the just previous network cycle. Let inv be the probability that a specific cache gets an invalidation from another. Then the probability x is:

$$x = \left(1 - \frac{\alpha(S,P)inv}{P}\right)^{P-1} \quad (9)$$

where $\alpha(S, P)$ is a function which predicts the average number of invalidated blocks by an invalidation. This function is, clearly, affected by the number of shared blocks and processors but is difficult to formulate since it should be derived from a given parallel program model. In this paper, we do not focus on the formulation but use an approximation for this function.

When a processor writes to a block in its cache, and if there is at least one other cache which contains the block as *clean*, the processor issues invalidations to those caches. Thus, inv is:

$$inv = \varphi_{normal}u(1-r)c + \varphi_{normal}u(1-H)d \quad (10)$$

The probability, w, that a processor falls into a waiting state to issue a request to the network is thought of as two cases: one is the case where the network is busy; the other is if the processor loses the network contention even if the network is not busy. In the former case, let $Q = Q_d + Q_c$ ($Q_d = \{Rc, Rd, Wc, Wd\}$, $Q_c = \{WcIV, WcIV, WB, Rp\}$) be a set of states in which a processor is issuing (data and coherence) requests to the network. If the state of a processor is in Q, the probability that it is acquiring the network and will not leave the state in the next network cycle is $(\eta_i - 1)/\eta_i$. Therefore, the probability $Busy$ that the network is busy is:

$$Busy = \binom{P-1}{1}\beta(1-\beta)^{P-2} \quad (11)$$

where $\beta = \sum_{i \in Q}(\eta_i - 1)/\eta_i$. In the latter case, the probability Win that a processor wins the network contention is:

$$Win = (1 - (1 - \varphi_{network})^P)\frac{1}{P\varphi_{network}} \quad (12)$$

Since the probability that a request is accepted at the cycle when there are issued requests is the reciprocal of the number of issued requests at the cycle. Thus the probability w that a processor falls into a waiting state is:

$$w = Busy + (1 - Busy)(1 - Win) \quad (13)$$

The request rate φ_i can be obtained as follows. By definitions, the normal request rate φ_{normal} is:

$$\varphi_{normal} = \frac{P_{COM}}{\eta_{COM}} \quad (14)$$

The memory request rate is equal to the sum of the ratio of leaving from the COM state with a cache miss and the ratio of leaving from the memory access waiting states. As described previously, the waiting state is divided in two cases; network contention and network busy . Let us refer to the former as the full waiting state and the latter as the residual waiting state. We assume that the processor issues a request per each waiting and each cycle in the full and residual waiting state respectively. Thus the probability, φ_{memory}, is:

$$\varphi_{memory} = ((1-u)(1-h) + u(1-H))\varphi_{normal} + Busy\sum_{i \in Q_d} P_i/\eta_i + (1 - Busy)\sum_{i \in Q_d} P_i \quad (15)$$

As is in the case of the memory request rate, the coherence request rate is:

$$\varphi_{coherence} = ((1-r)c + (1-H)d)\varphi_{normal} + m(1-x) + Busy \sum_{i \in Q_r} P_i/\eta_i + (1 - Busy) \sum_{i \in Q_r} P_i \quad (16)$$

In the above expression, $m(1-x)$ represents the request rate for writeback operations caused by invalidations.

Finally, the network request rate is:

$$\varphi_{network} = \varphi_{memory} + \varphi_{coherence} \quad (17)$$

The definition of Wt is the average time until the network is free, so Wt is:

$$Wt = \sum_{i \in Q} P_i \eta_i \quad (18)$$

The rate, y, of data requests to all requests (including writeback requests) from the COM state is:

$$y = \frac{\varphi_{normal}}{\varphi_{normal} + m(1-x)}. \quad (19)$$

3.5 The Procedure to Calculate the SMCI Model

To calculate the SMCI model, as described in subsection 2.1, we should start from appropriate initial values to get to a convergent point with regard to the network request rate. They should be as follows: $\varphi_{normal} = 1/\lambda$, $\varphi_{coherence} = 0$, $y = 1$, $x = 1$, $w = 1 - (1 - \varphi_{normal})^P$, $Wt = 1$.

4 Conclusion

In this paper we proposed a new analytic model, the SMCI model, for cache coherence based parallel computers by using a semi-markov process. To construct the SMCI model, we defined various processor request rates, showed the method of calculating the caching probability of a shared block through the shared block hit rate and the total number of shared blocks and gave the state definitions which can allow the same input parameters as used in Archibald's simulator. Thus the SMCI model could treat the cache coherence request as well as normal request with their waiting states, with considering cache replacement.

The current problem of the SMCI model is that we adopted rough approximation for the average number of invalidated blocks when an invalidation is issued. It should depend on parallel program models rather than architecture models. Therefore, its better approximation should be the future work.

参考文献

[1] Archibald, J. and Baer, J., "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," ACM Trans. on Computer Syst., vol.4, no.4, pp.273–298, 1986.

[2] Bhandarkar, D.P., "Analysis of memory interference in multiprocessors," IEEE Trans. Comput., vol.24, no.9, pp.897–908, 1975.

[3] Chiang, M. and Sohi, G., "Experience with Mean Value Analysis Models for Evaluating Shared Bus, Throughput-Oriented Multiprocessors," in Proc. ACM SIGMETRICS, pp.173–182, 1990.

[4] Joe, K. and Fukuda, A., "An Analytic Model for a Hierarchical Parallel System," in Proc. the 2nd Int'l workshop on Massive Parallelism: Hardware, Software and Applications, pp.287–304, 1994.

[5] Jõg, R., Vital, P.L. and Callister, J.R., "Performance Evaluation of a Commercial Cache-Coherent Shared Memory Multiprocessor," in Proc. ACM SIGMETRICS, pp.173–182, 1990.

[6] Kuck, D.J., Davidson, E.S., Lawrie, D.H. and Sameh, A.H., "Parallel Supercomputing Today and the Cedar Approach, Science," vol.231, no.2, pp.967–974, 1986.

[7] Lenoski, D.E., "THE DESIGN AND ANALYSIS OF DASH: A SCALABLE DIRECTORY-BASED MULTIPROCESSOR," Stanford Univ. PhDthesis, CSL-TR-92-507, 1992.

[8] Marsan, M.A. and Gerla, M., "Markov Models for Multiple Bus Multiprocessor Systems," IEEE Trans. Comput., vol.31, no.3, pp.239–248, 1982.

[9] McCredie, J.W., "ANALYTIC MODELS AS AIDS IN MULTIPROCESSOR DESIGN," in Proc. Ann. Princeton Conf. on Inf. Sci. and Sys., pp.186–191, 1973.

[10] Mori, S., Saito, H., Goshima, M., Yanagihara, M., Tanaka, T., Joe, K., Fraser, D., Nitta, H. and Tomita, S., "A distributed shared memory multiprocessor: Asura – memory and cache architectures–," in Proc. Supercomputing 93, pp.740–749, 1993.

[11] Mudge, T.N. and Al-Sadoun, H.B., "Memory Interference Models with Variable Connection Time," IEEE Trans. Comput., vol.33, no.11, pp.1033–1038, 1984.

[12] Mudge, T.N. and Al-Sadoun, H.B., "A semi-markov model for the performance of multiple-bus systems," IEEE Trans. Comput., vol.34, no.10, pp.934–942, 1985.

[13] Osaki, S., Applied Stochastic System Modeling, Springer-Verlag, 1992.

[14] Vernon, M., Lazowska, E.D. and Zahorjan, J., "An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols," in Proc. ISCA, pp.308–315, 1988.

[15] Wang, J. and Dubois, M., "Performance comparison of cache coherence protocols based on the access burst model," Computer Systems Science and Engineering, vol.5, no.3, pp.147–158, 1990.