Applying the Semi-Markov Memory and Cache Coherence Interference Model to an Updating Based Cache Coherence Protocol

Kazuki JOE and Akira FUKUDA

Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

In this paper, we propose the the method to apply the Semi-markov Memory and Cache coherence Interference (SMCI) model, which we had proposed for parallel computers with an invalidating based cache coherence protocol, to parallel computers with an updating based protocol. The proposed model, SMCI/Dragon, can be used for performance prediction of cache coherent parallel computers with the Dragon protocol as well as the original SMCI model. Conventional analytic models by a stochastic process for parallel computers have an unavoidable disadvantage of explosions of the number of states as the system size is enlarged even if they are for simple parallel computers without cache coherence mechanisms. The number of states required by constructing the SMCI model, however, does not depend on the system size but only on the kind of cache coherence protocol. The number of states for the Dragon protocol is only 20 as is described in this paper.

SMCI モデルのアップデート型キャッシュ・コヒーレンス・プロトコルへの適用

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

本稿では、既に我々の提案している、インバリデーション型キャッシュ・コヒーレンス・プロトコルを採用した並列計算機に対する解析モデル、Semi-markov Memory and Cache coherence Interference (SMCI) モデルの、アップデート型キャッシュ・コヒーレンス・プロトコルへの適用について報告する。本稿で提案する SMCI/Dragon モデルは、元の SMCI モデル同様、代表的なアップデート型プロトコルである Dragon を採用した並列計算機の性能予測を行なう。確率過程等を利用したこれまでの解析モデルでは、並列システムの大規模化に伴い、状態数の爆発的増化という避けがたい問題があった。SMCI モデルでは、状態数はシステムの規模に無関係に定まる。本稿で報告する Dragon プロトコルに対する状態数は２０である。

1 Introduction

Various technologies have been investigated for improving parallel computer architectures, such as cache coherence protocols [1], processor clustering schemes [5] or scalable shared memory mechanisms [7].

It is quite useful to predict the performance of such large scaled and complicated parallel computer architectures in advance of their detail designs. One of such methodologies is simulation. However, simulating large scaled parallel computers leads to another problem of the computational cost. For example, we can not execute any simulation program on a typical workstation which simulates a parallel computer with 1,024 processors.

Analytic modeling is widely known as a valid and inexpensive methodology for the evaluation of parallel computer systems. The problem is that most of previous works on analytic modeling, which were mostly by using of a markov chain [2] or queueing theory [6], do not support parallel computers with cache memory.

Using stochastic models, some work focused on cache coherence mechanisms [8] but did not support network contention nor memory interference which takes place in parallel computers.

We, therefore, have proposed the Semi-markov Memory and Cache coherence Interference (SMCI) model [3], which can analyze the Synapse [1] cache coherence protocol mechanism as well as network contention and memory interference both for data and cache management requests of shared memory parallel computers. The advantage of the SMCI model is its small number of states. When the SMCI model is applied to the parallel computer with thousands of processors which supports the Synapse cache coherence protocol, the number of states is only 19. Because of this small number of states, the SMCI model can succeed to predict the performance of a given parallel computer architecture within 300 micro seconds on a typical workstation.

The SMCI model had two major problems of its application to 1) larger and, therefore, more complicated parallel computer architectures and to 2) other cache coherence protocols.

On concerning the former, we have already proposed the Hierarchical SMCI model [4], which can be used for a scalable shared memory parallel computer. The target architecture is based on a processor clustering structure with hierarchical memory system connected by a hierarchical network: The processor cluster consists of several processors, their own private caches and local memory connected by a bus network. It also has a gateway module to communicate with other processor clusters. The gateway module contains of global cache and part of distributed global memory. Each processor cluster is connected by a global network via its gateway module. The SMCI model was extended to be applied to this large and complicated architecture. We introduced the hierarchy into the model in order to be adapted to the hierarchical system, and constructed the Hierarchical SMCI model which consists of two SMCI models, one for representing the processor and the other for representing the processor cluster.

As for the latter, we should demonstrate that the SMCI model can be applied to other cache coherence protocols than Synapse which is based on an invalidating policy and a memory-to-cache transfer mechanism. For example, we should indicate that the SMCI model can be also constructed for the parallel computer with a cache coherence protocol which has an opposite characteristics to the Synapse protocol.

Thus we propose, in this paper, the method for applying the SMCI model to parallel computers with a cache coherence protocol of an updating policy and a cache-to-cache transfer mechanism. Such a cache coherence protocol is known to be the Dragon protocol [1]. Although the key concept of the SMCI/Dragon model is same to that of the SMCI/Synapse model, they have slightly different state definitions with each other, and derivation of internal variables such as various request rates is also different.

2 Preliminary

2.1 Policy of Modeling

Using the same concept as the SMCI/Synapse model, we first formulate the relation between the limiting probabilities of states of a processor and processor request rates. Giving initial values to the processor request rates, the limiting probabilities can be calculated. Based on them, the processor request rates are updated. Repeating these calculations, we can find a convergent point for a request rate which indicates the steady states of the system.

2.2 Assumptions of the Model

The target parallel computer consists of several processors, their own private caches and memory connected by a single shared bus network. The cache coherence mechanism with the Dragon protocol [1] is supported by hardware.

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

1. The behavior of processors can be modeled as identical stochastic processes.
2. The duration of issueing and receiving requests are independent, and expressed as geometrically distributed discrete random variables with mean λ_{nor} (issueing a request), λ_{CT} (receiving requests) and λ_{Upd} (receiving requests) respectively. Request service times are given determinately by a hardware specification of the system.
3. The running program is in a steady state; the initial paging for loading or the initial poor cache hit rate is not considered. But the replacement of cache blocks is considered.

2.3 Semi-Markov Process

In this paper, a Semi-Markov Process (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}. See [3] for more details.

2.4 Dragon Protocol

Dragon [1] is a cache coherence protocol of which characteristics is represented as a broadcast request of updating other caches at a write operation and as a cache-to-cache transfer when a cache miss occurs with other

表 1: Summary of Dragon Protocol (CCW: Cache-to-Cache tran. of Word, MCB: Memory-to-Cache tran. of Block)

	cache	other caches	bus	memory	remark
Write Hit	CE,DE →DE	not exist	no use	no use	no use of a bus
	CS →DS	CS,DS →CS	CCW	no use	broadcast of updating
	DS →DS	CS →CS	CCW	no use	broadcast of updating
Read Miss	→CE	not exist	MCB	Load	read from memory
	→CS	CE,CS →CS	CCW	no use	cache-to-cache transfer
	→CS	DE,DS →DS	CCW	no use	cache-to-cache transfer
Write Miss	→DE	not exist	MCB	Load	read from memory then write to cache
	→DS	CE,CS,→CS DE,DS	CCW	no use	cache-to-cache transfer then write to cache, broadcast of updating
Replace	DE,DS →	no change	CMB	Store	write back to memory
	CE,CS →	no change	no use	no use	the block is flushed

cache's having a copy of the block. In the Dragon protocol, a data block can be in one of four states : 1) Clean-Exclusive (**CE**, only copy in caches, but not modified), 2) Clean-Shared (**CS**, not modified, possibly other caches with a copy), 3) Dirty-Exclusive (**DE**, only copy in caches and modified) and 4) Dirty-Shared (**DS**, modified, possibly other caches with a copy in CS). Table 1 summarizes the work of Dragon.

表 2: State definitions in the SMCI/Dragon model

COM	**COM**putation
Rh	**R**ead **h**it
WhE	**W**rite **h**it on a clean/dirty **E**xclusive block
WhS	**W**rite **h**it on a clean/dirty **S**hared block
WhB	**B**roadcast of upd. caused by a **W**rite **h**it
$\overline{WhB}$	Waiting state for WhB
Rm	**R**ead from **m**emory
$\overline{Rm}$	Waiting state for Rm
Rc	**R**ead from another **c**ache
$\overline{Rc}$	Waiting state Rc
Wm	**W**rite to **m**emory
$\overline{Wm}$	Waiting state Wm
Wc	**W**rite to a block loaded from another **c**ache
$\overline{Wc}$	Waiting state Wc
WcB	**B**roadcast of updating caused by Wc
$\overline{WcB}$	Waiting state WcB
Rep	**Rep**lacement caused by Rm, Rc, Wm, Wc
$\overline{Rep}$	Waiting state Rep
Flu	**Flu**sh caused by Rm, Rc, Wm and Wc
CT	Loading a block to a network caused by a request of **C**ache-to-cache **T**ransfer
Upd	**Upd**ating a block caused by a broadcast request of updating

2.5 Definitions of Various Request Rates

In a parallel computer, requests from a processor are determined at run time. They consist of original requests from a given program and re-issued requests from processor waiting states because of network contention. The former might be bound for cache or memory. If they are accepted at private cache, the requests do not affect the network. Furthermore, there is a request for replacement of a cache entry which affects the network performance but is not a normal data request.

To construct the SMCI/Synapse model, we have proposed new definitions for various request rates [3] to formulate the above phenomena. In this paper, we propose similar kinds of request rates for construction of the SMCI/Dragon model: a) **Normal request rate** φ_{nor} is the rate that a processor issues a request just after its computation state, b) **Memory request rate** φ_{mem} is the rate that a processor issues a request to memory either from cache miss, replacement or waiting states, c) **Cache-to-cache communication request rate** φ_{cc} is the rate that a processor issues a cache-to-cache communication request either from cache miss/hit or waiting states and d) **Network request rate** φ_{net} is the rate that a processor issues a request to the network and is expressed as the sum of φ_{mem} and φ_{cc}.

3 The SMCI Model

3.1 Definition of States

To construct the SMCI/Dragon model, we first define each state of a processor. Table 2 shows state definitions of a processor to its cache, memory and network based on the Dragon protocol. As shown in the table, it requires only 20 states to represent the possible behavior of a processor in the Dragon protocol. Table 3 indicates several sets of the states, which are used for derivation of internal variables of the model.

3.2 State Transitions

Figure 1 shows transitions between the defined states. A state transition occurs when any kinds of request is issued, completed or arrived. 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, t represents the probability that no other caches have a copy of the requested block, w represents the probability that the request is not accepted, u represents the probability that the processor generates a reference to a shared block and m represents the probability that the replacing block has been already modified. There are three kinds of state transition from COM; 1) caused by issueing data requests such as read or write, 2) caused by a cache-to-cache transfering request and 3) caused by a broadcast request of updating. y represents the rate of 1) to all transitions and z represents the rate of 2) to the sum of 2) and 3).

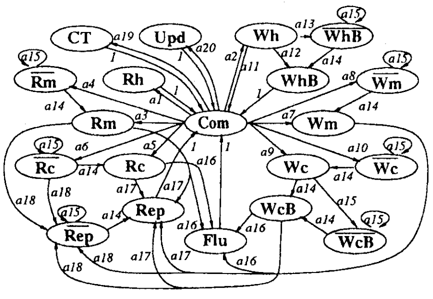
While t is derived from input parameters, w, y and z are derived from the limiting probability and various request rates. All other variables are given as input parameters.

The COM state changes in two cases; 1) the processor issues a data request and 2) the processor responds to a request from another processor. In the latter case, the processor's behavior is to load a cache block to a network responding to a cache-to-cache transfer request from another processor (CT) or to update a cache block responding to a broadcast request from another processor (Upd). From these states, they change to COM directly. In the former case, it changes to Rh, Rm, Rc, $\overline{Rm}$, $\overline{Rc}$, WhE, WhS, Wm, Wc, $\overline{Wm}$ or $\overline{Wc}$, depending on whether the request is read or write, whether a cache hit occurs, whether the requested block is in the dirty/clean exclusive or shared when a write hit occurs, whether the request requires a memory access or a cache-to-cache transfer, or whether the network is busy respectively. Rh and WhE change only to COM. Remark that in Rc and Wc, the requested data block is transfered from another cache which has a copy of the block. Wc changes to WcB or $\overline{WcB}$ to issue a broadcast request of updating. Similarly WhS changes to WhB or $\overline{WhB}$ for broadcasting. A state of cache miss (Rm, Rc, Wm or WcB) requires replacement of a cache block by assumption. The replaced cache block must be written back to memory (Rep or $\overline{Rep}$) or simply discarded (Flu). The waiting states, $\overline{WhB}$, $\overline{Rm}$, $\overline{Rc}$, $\overline{Wm}$, $\overline{Wc}$, $\overline{WcB}$ and $\overline{Rep}$, change to WhB, Rm, Rc, Wm, Wc, WcB and Rep respectively or change 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 follows. When a request occurs in COM, it must be a data request (with the rate y) or caused by a request (of cache-to-cache transfer or broadcast of updating) from another processor (with the 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 can be satisfied at the cache. When the request is bound for a private block (with the probability $1-u$), the private block cache hit rate is h. Otherwise it is bound for a shared block (with the 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 probabilities because of lack of space.

3.3 Construction of an SMP

We have discussed definitions of processor's states and transition probabilities between the defined states. Let us investigate distributions of sojourn times for each state. States of a processor can be classified into three groups with regard to distributions of their sojourn times: 1) a computation state, 2) request service states for cache or memory and 3) waiting states. As described in Sect.2.2, we assume that the sojourn time for computation is geometrically distributed. Strictly speaking, each transition from COM has its own distribution of the sojourn time according to the next state (Rh, Rm, $\overline{Rm}$, Rc, $\overline{Rc}$, WhE, WhS, Wm, $\overline{Wm}$, Wc, $\overline{Wc}$, CT, Upd). It is, however, extremely difficult to obtain such distribution functions. Therefore we assume three kinds of geometrically dis-



a1=y(h(1-u)+H u)r , a2=y(h(1-u)+H u)(1-r) , a3=y((1-h)(1-u)+(1-H)ut)r(1-w) ,
a4=y((1-h)(1-u)+(1-H)ut)rw , a5=y(1-H)ur(1-t)(1-w) , a6=y(1-H)u(1-t)w ,
a7=y((1-h)(1-u)+(1-H)ut)(1-r)(1-w) , a8= y((1-h)(1-u)+(1-H)ut)(1-r)w ,
a9=y(1-H)u(1-r)(1-t)(1-w) , a10=y(1-H)u(1-r)(1-t)w , a11=t , a12=(1-t)(1-w) ,
a13=(1-t)w , a14=1-w , a15=w , a16=1-m , a17=m(1-w) , a18=mw , a19=yz , a20=y(1-z)

図 1: State transition graph of an SMP based on the Dragon protocol

tributed functions for the sojourn times of computation; 1)parameter λ_{nor} for transitions caused by issueing data requests, 2) parameter λ_{CT} for a transition caused by a cache-to-cache transfering request and 3) parameter λ_{Upd} for a transition caused by a broadcast request of updating. The sojourn times for the request service states are determinately given by the below function $F_{ij}()$.

$$F_{ij}(t) = \begin{cases} 1 & \text{if } t \geq \text{State } i \text{ Serv. Time } (i \in Q_{cache} \cup Q_{net}) \\ 0 & \text{otherwise} \end{cases}$$

For example, the access times for a cache read, a memory read and a broadcast request are 1, 16 and 4 cycles respectively. These values are determined by a specification of the target architecture. Furthermore, we assume that the sojourn time for waiting states is geometrically distributed with parameter Wt as 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 times of each state of X are given by the above time distributions. Considering each sojourn time as random variable sequences $T = \{T_n\}_{n=1}^{\infty}$, 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 obtain the limiting probability of the SMP. As described in Sect.2.1, we start from appropriate initial request rates, obtain the limiting probabilities, update the request rates, obtain the new limiting probabilities and repeat these calculations until the request rates are saturated. Thus we obtain the steady states of the system.

3.4 Limiting Probabilities and Request Rates

To calculate various request rates and limiting probabilities, several appropriate input parameters are needed. They are the number of processors N, private block cache hit rate h, shared block access rate$u$, read request rate N, the number of shared blocks in a given program E and the probability that the replacing block has been already modified m, which are the same ones as in Archibald's simulation. Also each sojourn time for COM (λ_{nor}) and request service states (η_i, $i \in Q_{cache} \cup Q_{net}$) is given as input parameters. Remark that λ_{nor} is the distribution parameter of transitions from COM caused by issueing data requests.

The aim of this subsection is to derive internal variables by input parameters. The internal variables are summarized in Table 4.

表 3: Sets of states of the SMCI/Dragon model

Q	A set of all states
Q_{cache}	$= \{Rh, WhE, WhS, CT, Upd, Flu\}$
Q_{m_req}	$= \{Rm, Wm, Rep\}$
$Q_{m\_req\_w}$	$= \{i \vert i \in Q_{m\_req}\}$
Q_{c_req}	$= \{Rc, Wc, WhB, WcB\}$
$Q_{c\_req\_w}$	$= \{i \vert i \in Q_{c\_req}\}$
Q_{net}	$= Q_{m\_req} \cup Q_{c\_req}$
Q_{net_w}	$= \{i \vert i \in Q_{net}\}$

表 4: Internal Variables used in the SMCI/Dragon Model

H	shared block cache hit rate
Ψ	expected number of shared blocks in a cache
t	probability that no other caches have a copy of the requested block
w	probability that the request to a network is not accepted
y	rate of transitions from COM caused by data requests to all transitions from COM
z	rate of transitions from COM caused by cache-to-cache transfer requests to those by all transitions except data requests
Wt	average sojourn time of waiting states
λ_{CT}	distribution parameter of transitions from COM caused by cache-to-cache trans. req.
λ_{Upd}	distribution parameter of transition from COM caused by broadcast req. of upd.
φ_{nor}	normal request rate
φ_{mem}	memory request rate
φ_{cc}	cache-to-cache communication request rate
φ_{net}	network request rate
$\{\pi_i\}$	stationary distribution of the EMC
$\{P_i\}$	limiting probability of the SMP

In [3], we denoted that H was derived from [8]. Since no analytical result of H for the Dragon protocol has been obtained, we approximate H to that for the Write-once, the Illinois and the Berkeley protocols (They have the same value.) in [8] as below.

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

where ls represents the access burst length[8].

ψ was given in [3] by using H and E.

The feature of the Dragon protocol is a cache-to-cache transfer instead of a memory access in the case that a cache miss occurs and other caches have a copy of the block. To formulate this mechanism, we introduce the probability t that no other caches have a copy of the requested (shared) block as below.

$$t = \left(1 - \frac{\Psi}{E}\right)^{N-1} \tag{2}$$

The probability, w, that a processor falls into a waiting state to issue a request to the network is derived in the same way as in [3]. There are two cases for such waiting states: one is the case the network is busy and the other is the case the processor loses network contention even if the network is not busy. In the former case, the probability that another processor is accessing cache (cache-to-cache transfers or broadcasting) or memory and will not leave the state in the next network cycle is $P_i\frac{\eta_i - 1}{\eta_i}$ ($i \in Q_{c\_req} \cup Q_{m\_req}$). Therefore, the probability $Busy$ that the network is busy is given by

$$Busy = \begin{pmatrix} N-1 \\ 1 \end{pmatrix} \beta(1-\beta)^{N-2} \tag{3}$$

where $\beta = \sum_{i \in Q_{c\_req} \cup Q_{m\_req}} P_i\frac{\eta_i - 1}{\eta_i}$. In the latter case, the probability Win that the processor wins the network contention is defined as

$$Win = (1-(1-\varphi_{net})^N)\frac{1}{N\varphi_{net}}. \tag{4}$$

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 can be expressed as

$$w = Busy + (1 - Busy)(1 - Win). \tag{5}$$

The request rate φ_i can be obtained as below. By definitions, the normal request rate φ_{nor} is given by

$$\varphi_{nor} = \frac{1}{\lambda_{nor}}. \tag{6}$$

The memory request rate is equal to the sum of the rate of leaving from the COM state to Q_{m_req} and the rate of leaving from $Q_{m\_req\_w}$. As described previously, a 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 a processor issues a request per each waiting and each cycle in the full and residual waiting state respectively. Thus the probability, φ_{mem}, is defined by

$$\begin{aligned} \varphi_{mem} = & ((1-h)(1-u)+(1-H)tu)\,\varphi_{nor} \\ + & \ m\,((1-h)(1-u)+(1-H)u)\,\varphi_{nor} \\ & Busy \sum_{i \in Q_{m\_req\_w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{m\_req\_w}} P_i. \end{aligned} \tag{7}$$

As is in the case of the memory request rate, the cache-to-cache communication request rate is obtained from

$$\begin{aligned} \varphi_{cc} = & (1-H)(1-t)u\varphi_{nor} + u(1-r)(1-t)\varphi_{nor} \\ + & \ Busy \sum_{i \in Q_{c\_req\_w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{c\_req\_w}} P_i. \end{aligned} \tag{8}$$

The network request rate is expressed as

$$\varphi_{net} = \varphi_{mem} + \varphi_{cc} \tag{9}$$

The definition of Wt is the average time until the network is free, thus Wt is given as below.

$$Wt = \frac{\sum_{i \in Q_{net\_w}} P_i \eta_i}{\sum_{i \in Q_{net\_w}} P_i} \tag{10}$$

The rate y of transitions from COM caused by data requests to all transitions from COM is derived from the cache-to-cache communication request φ_{cc}. As describe above, φ_{cc} consists of the cache-to-cache transfer request rate $(1-H)(1-t)u\varphi_{nor}$, the updating broadcast request rate $u(1-r)(1-t)\varphi_{nor}$ and the request rate from their waiting state. Thus the probability z_{CT} that a processor receives a cache-to-cache transfer request and the probability z_{Upd} that a processor receives a broadcast request of updating are given by

$$z_{CT} = 1 - \left(1 - \frac{\alpha(N,E)}{N}(1-H)(1-t)u\varphi_{nor}\right)^{N-1} \tag{11}$$

$$z_{Upd} = 1 - \left(1 - \frac{\alpha(N,E)}{N}u(1-r)(1-t)\varphi_{nor}\right)^{N-1} \tag{12}$$

respectively where $\alpha()$ is the function[1] which predicts the average number of caches with having a copy of the requested block.

Therefore y is expressed as below.

$$y = \frac{\varphi_{nor}}{1-(1-\varphi_{nor})(1-z_{CT})(1-z_{Upd})} \tag{13}$$

Similarly, the rate z of transitions from COM caused by cache-to-cache transfer requests to those by all transitions except data requests is obtained from

$$z = \frac{z_{CT}}{(1-(1-z_{CT})(1-z_{Upd}))}. \tag{14}$$

The time distribution parameter λ_{CT} of transitions from COM caused by cache-to-cache transfer requests and λ_{Upd} of transitions by broadcast requests of updating are defined as reciprocals of z_{CT} and z_{Upd} respectively.

Finally, the average sojourn time of COM can be obtained as below.

$$\eta_{COM} = y\lambda_{nor} + (1-y)z\lambda_{CT} + (1-y)(1-z)\lambda_{Upd} \tag{15}$$

3.5 Initial Values for Calculations

To obtain the converged limiting probabilities of the SMCI/Dragon model, as described in subsection 2.1, we start from appropriate initial values. An example of sets of initial values are: $\varphi_{nor} = \frac{1}{\lambda_{nor}}$, $\varphi_{mem} = (1-h)\varphi_{nor}$, $\varphi_{cc} = 0$, $y = 1.0$, $z = 0.0$, $w = 1-(1-\varphi_{net})^N$ and $Wt = 1$.

[1] 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.

4 Conclusion

In this paper we proposed the method to apply the SMCI model to parallel computers with a cache coherence protocol of an updating policy and a cache-to-cache transfer mechanism, and obtained the SMCI/Dragon model. To construct the SMCI/Dragon model, we re-defined various processor request rates to suit the Dragon protocol and gave the state definitions which can allow the same input parameters as used in Archibald's simulation. Thus the SMCI/Dragon model could treat the cache-to-cache communication request as well as the normal memory request with their waiting states, with considering the cache replacement.

The current problem of the SMCI/Dragon model is that we adopted rough approximation for the average number of caches with having a copy of the requested blocks. 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] James Archibald and Jean-Loup Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Trans. on Computer System*, 4(4):273–298, 1986.

[2] Dileep P. Bhandarkar. Analysis of memory interference in multiprocessors. *IEEE Trans. on Computer*, C-24(9):897–908, 1975.

[3] K. Joe and A. Fukuda. Analytic modeling of cache coherence based parallel computers. Technical Report 95-MPS-3-3, IPSJ SIGMPS, 1995.

[4] Kazuki Joe and Akira Fukuda. An analytic model for a hierarchical parallel system. In *International Workshop on Massive Parallelism: Hardware, Software and Applications*, pages 287–304, 1994.

[5] David J. Kuck, Edward S. Davidson, Duncan H. Lawrie, and Ahmed H. Sameh. Parallel supercomputing today and the cedar approach. *Science*, 231(2):967–974, 1986.

[6] John W. McCredie. Analytic models as aids in multiprocessor design. In *Ann. Princeton Conference on Information Science and System*, pages 186–191, 1973.

[7] S. Mori, H. Saito, M. Goshima, M. Yanagihara, T. Tanaka, K. Joe, D. Fraser, H. Nitta, and S. Tomita. A distributed shared memory multiprocessor: Asura – memory and cache architectures–. In *Supercomputing 93*, pages 740–749, 1993.

[8] Jin-Chin Wang and Michel Dubois. Performance comparison of cache coherence protocols based on the access burst model. *Computer System Science and Engeneering*, 5(3):147–158, 1990.