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

Kazuki Joe and Akira Fukuda

Nara Institute of Science and Technology, JAPAN
{joe,fukuda}@is.aist-nara.ac.jp

Abstract. In this paper, we propose a 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 an updating based protocol. The proposed model, SMCI/Dragon, predicts the performance of cache coherent parallel computers with Dragon protocol with extremely inexpensive computational costs. Using the extended model, we investigate several comparative experiments with widely known simulation. The results show that there are only 5.4% differences.

1 Introduction

With given hardware configurations and/or executions of given parallel programs, a criterion for performance evaluation of parallel computers may be processor utilization, network utilization or the frequency of overhead which is needed to keep shared data consistent. Such performance evaluation can be done by address trace simulation besides actual measurements.

Predictive performance evaluation of given parallel computer architectures and executions of given parallel programs in advance of their implementations is extremely difficult, since the total number of predictions explosively increases as the number or the range of parameters, which represents the size and complexity of architectures or the method of parallelization, increases.

While simulation is an effective methodology for the performance evaluation of architecture models with the given small number of parameters, it is apparently inefficient to apply the simulation to the explosive number of parameter choices because of computational costs.

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 [5], do not support parallel computers with cache memory. Using stochastic models, some work focused on cache coherence mechanisms [6] 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 [4], which can analyze Synapse [1] cache coherence

protocol 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. Even if the SMCI model is applied to a parallel computer with thousands of processors which has 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 performance of given parallel computer architectures with the order of microseconds execution time 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 [3], which can be used for a scalable shared memory parallel computer[2]. The target architecture is based on a processor clustering structure with hierarchical memory (local/global cache and memory) system connected by a hierarchical network. The SMCI model was extended to be applied to this large and complicated architecture as the Hierarchical SMCI model.

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 parallel computers with a cache coherence protocol which has the opposite characteristics to Synapse.

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. Dragon[1] is one of such cache coherence protocols. 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. Validation of the SMCI/Dragon model is shown by the comparison with results of Archibald's simulation [1].

The structure of this paper is as follows. Assumptions for the SMCI/Dragon model and definitions of various request rates are described in Sect.2. State definitions and derivation of internal variables are shown in Sect.3. The performance evaluation and its comparison with simulation results are given in Sect.4.

2 Preliminary

As the same concept to the SMCI/Synapse model, we first define the states of a processor with giving transitions probabilities among the states to construct a stochastic process under the below assumption. Then we formulate the relation between the limiting probabilities of states 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.

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 Dragon is supported by hardware. Application programs which run on given architecture models are assumed to be an SPMD (Single Program Multiple Data).

The architecture configuration and the software characteristics of given programs are expressed by parameters described in Sec.3.3.

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

- 1) The behavior of each processor is independent and can be modeled as an identical stochastic process.
- 2) The duration of issuing and receiving requests is independent, and expressed as geometrically distributed discrete random variables. Request service times are given determinately by architecture parameters.
- 3) The running program is in a steady state; initial paging for loading or initial poor cache hit rate is not considered. But the replacement of cache blocks is considered.

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} . The average sojourn time η_i is expressed as $\eta_i = \sum_{j=1}^K p_{ij} \sum_{n=0}^{\infty} t_n F_{ij}(t_n)$.

If an SMP has an ergodic irreducible embedded markov chain (EMC), the limiting probability of state i can be obtained from $P_i = \pi_i \eta_i / \sum_{j=1}^K (\pi_j \eta_j)$ where $\{\pi_i\}$ is the stationary distribution of an EMC.

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. Furthermore, there are cache coherence requests, which affect network performance but not normal data requests.

To construct the SMCI/Synapse model, we have proposed new definitions of various request rates [4] to formulate the above phenomena. In this paper, we propose similar kinds of request rates for the SMCI/Dragon model: a) **Normal request rate** φ_{nor} is the rate that a processor issues a data 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 misses, replacements 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 misses/hits or waiting states and d) **Network request rate** φ_{net} is the rate that a processor issues a request to a network and is expressed as the sum of φ_{mem} and φ_{cc} .

3 The SMCI/Dragon Model

3.1 Definition of states and their transitions

To construct the SMCI/Dragon model, we first define each state of a processor. The definition is as follows: 1) *COM* $\equiv$ computation, 2) *Rh* $\equiv$ cache read hit,

Fig. 1. State transition graph of an SMP based on Dragon

3) WhE $\equiv$ write hit on a clean/dirty exclusive block, 4) WhS $\equiv$ write hit on a clean/dirty shared block, 5) WhB $\equiv$ broadcast of updating caused by WhS , 6) $\overline{WhB}$ $\equiv$ its waiting state, 7) Rm $\equiv$ read from memory, 8) $\overline{Rm}$ $\equiv$ its waiting state, 9) Rc $\equiv$ read from another cache, 10) $\overline{Rc}$ $\equiv$ its waiting state, 11) Wm $\equiv$ write to memory, 12) $\overline{Wm}$ $\equiv$ its waiting state, 13) Wc $\equiv$ write to a block loaded from another cache, 14) $\overline{Wc}$ $\equiv$ its waiting state, 15) WcB $\equiv$ broadcast of updating caused by Wc , 16) $\overline{WcB}$ $\equiv$ its waiting state, 17) Rep $\equiv$ replacement caused by cache misses, 18) $\overline{Rep}$ $\equiv$ its waiting state, 19) Flu $\equiv$ flush caused by cache misses, 20) CT $\equiv$ loading a block to a network caused by a request of cache-to-cache transfer and 21) Upd $\equiv$ updating a block caused by broadcast requests. Thus it requires only 21 states to represent the possible behaviors of a processor.

For simple derivation of internal variables in Sec.3.3, several sets of states are defined here: Q = the set of all states, $Q_{cache} = \{Rh, WhE, WhS, CT, Upd, Flu\}$, $Q_{m-req} = \{Rm, Wm, Rep\}$, $Q_{m-req-w} = \{\bar{i} | i \in Q_{m-req}\}$, $Q_{c-req} = \{Rc, Wc, WhB, WcB\}$, $Q_{c-req-w} = \{\bar{i} | i \in Q_{c-req}\}$, $Q_{net} = Q_{m-req} \cup Q_{c-req}$, $Q_{net-w} = \{\bar{i} | i \in Q_{net}\}$.

Figure 1 shows transitions between the defined states. A state transition occurs when any kind of request is issued, completed or arrived¹. We do not explain each state transition probability because of lack of space. See [4] for their details.

3.2 Construction of an SMP

We have discussed definitions of processor's states and transitions probabilities between the defined states. Let us investigate distributions of sojourn times for the states. 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,

¹ An invalidation arrives at a processor.

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. It is, however, extremely difficult to obtain such distribution functions. Therefore we assume three kinds of geometrically distributed functions for the sojourn times of computation; 1) parameter λ_{nor} for transitions caused by issuing data requests, 2) parameter λ_{CT} for a transition caused by a cache-to-cache transferring 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 function $F_{ij}(t)$ which returns 1 if $t \geq$ the access time of state i ($i \in Q_{cache} \cup Q_{net}$) or 0 otherwise.

Furthermore, we assume that the sojourn time for waiting states is geometrically distributed with parameter Wt as given later.

The stochastic process $X = \{X_n\}_{n=1}^{\infty}$ defined by the above state space and transitions probabilities represents the behavior of a processor. It is trivial that X is a markov chain because of assumptions in Sect.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 use the second equation of Sec.2 to get the limiting probability of the SMP.

3.3 Limiting probabilities and request rates

To calculate various request rates and limiting probabilities, several appropriate input parameters, which describe the architecture configuration and the software characteristics of given programs, are needed. The architecture parameters are N (number of processors) and η_i (request service time for cache and memory: $i \in Q_{cache} \cup Q_{net}$). The software parameters are h (private block cache hit rate), r (read request rate), u (shared block access rate), E (number of shared blocks in a given program), m (probability that the replacing block has been already modified) and λ_{nor} (distribution parameter of transitions from *COM* caused by issuing data requests).

The aim of this subsection is to derive internal variables by input parameters. The internal variables are as follows: 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_{Upd} (distribution parameter of transitions from *COM* caused by cache-to-cache transfer requests) and λ_{Upd} (distribution parameter of transitions from *COM* caused by broadcast requests of updating).

In [3, 4], we denoted that H was derived from [6]. Since no analytical result of H for Dragon has been obtained, we approximate H to that for the Write-once, the Illinois and the Berkeley protocols (They have the same value.) in [6]. The

expected number of shared blocks in a cache, ψ , was given in [4] 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 $t = (1 - \Psi/E)^{N-1}$.

The probability w is derived in the same way as in [4]. There are two cases for such waiting states: 1) the network is busy and 2) the processor loses network contention even if the network is not busy. In the former case, the probability that another processor is accessing the 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 $Busy = \binom{N-1}{1} \beta (1 - \beta)^{N-2}$ 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 $Win = (1 - (1 - \varphi_{net})^N) / (N \varphi_{net})$. Since the probability that a request is accepted at a cycle when there are issued requests is the reciprocal of the number of issued requests at the cycle. Thus the probability w can be expressed as $w = Busy + (1 - Busy)(1 - Win)$.

By definitions, the normal request rate is given by $\varphi_{nor} = 1/\lambda_{nor}$.

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. We assume that the processor issues a request again in the next cycle in the former case while it does again with delayed Wt cycle (after waiting the average sojourn time of the waiting state) in the latter case. The cache-to-cache communication request rate is obtained in the same way.

$$\begin{aligned} \varphi_{mem} &= ((1-h)(1-u) + (1-H)tu)\varphi_{nor} + m((1-h)(1-u) \\ &\quad + (1-H)u)\varphi_{nor} + Busy \sum_{i \in Q_{m_req-w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{m_req-w}} P_i \\ \varphi_{cc} &= (1-H)(1-t)u\varphi_{nor} + u(1-r)(1-t)\varphi_{nor} \\ &\quad + Busy \sum_{i \in Q_{c_req-w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{c_req-w}} P_i \end{aligned}$$

The definition of Wt is the average time until the network is free, thus Wt is given as $Wt = \left(\sum_{i \in Q_{net-w}} P_i \right) / \left(\sum_{i \in Q_{net-w}} \frac{P_i}{\eta_i} \right)$.

The rate y is derived from φ_{cc} which consists of the cache-to-cache transfer request rate, the updating broadcast request rate 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 $z_{CT} = 1 - (1 - \frac{\alpha(N,E)}{N}(1-H)(1-t)u\varphi_{nor})^{N-1}$ and $z_{Upd} = 1 - (1 - \frac{\alpha(N,E)}{N}u(1-r)(1-t)\varphi_{nor})^{N-1}$ respectively, where $\alpha()$ is the

function² which predicts the average number of caches with having a copy of the requested block. Therefore y is expressed as $y = \varphi_{nor} / (1 - (1 - \varphi_{nor})(1 - z_{CT})(1 - z_{Upd}))$. Similarly, the rate z is obtained from $z = z_{CT} / (1 - (1 - z_{CT})(1 - z_{Upd}))$. The time distribution parameters λ_{CT} and λ_{Upd} are defined as reciprocals of z_{CT} and z_{Upd} respectively.

Finally, the average sojourn time of *COM* can be obtained by using the first equation of Sec.2 as $\eta_{COM} = y\lambda_{nor} + (1 - y)z\lambda_{CT} + (1 - y)(1 - z)\lambda_{Upd}$.

3.4 Initial values for calculations

To obtain the converged limiting probabilities of the SMCI/Dragon model, as described in subsection 2, we start from appropriate initial values. An example of sets of initial values are: $\varphi_{mem} = (1 - h)\varphi_{nor}$, $\varphi_{cc} = 0$, $y = 1.0$, $z = 0.0$, $w = 1 - (1 - \varphi_{net})^N$ and $Wt = 1$.

4 Evaluation

To validate the SMCI/Dragon model, we did the same experiments as Archibald's simulation [1] with the same system model. The input parameters are exactly same to experiments of Archibald's simulation. As for the system model, the sojourn times for request service states are given; 1 for a cache service, 4 for a broadcast and 16 for a memory service. And the distribution parameter of transitions from *COM* caused by data requests is 3 which is consistent with Archibald's workload. We also approximate the function $\alpha()$, which predicts the average number of caches with having a copy of the requested blocks, by simply $\frac{N}{2}$ for these experiments.

In these comparative experiments, it turned out that all results produced by the SMCI/Dragon model were good approximations to simulation results. And the average percentage difference between the SMCI/Dragon model and Archibald's simulation was only 5.4%. The accuracy of the SMCI/Dragon model is well enough that it can predict the relative relation between input parameters and the system performance. Thus, our main goal was achieved since we have interests in analyses of the influence of various architecture and software parameters to the system performance.

As for computation costs, the SMCI/Dragon model requires only 50 mil seconds to do all the experiments on a desk-top workstation while Archibald's simulation required 25,000 system cycle runs for each experiment³. The elapsed time of simulation of a parallel computer increases as the number of processors increase while the elapsed time of the SMCI/Dragon model does not increase as the number of processors. The computation of the steady state of an SMP,

² We do not focus on the formulation of this function but use an approximation.

³ Although [1] does not describe the elapsed time of those experiments, it can be estimated at several hours since our simulator requires about two hours to perform those experiments on the same workstation.

which accounts for most part of computation, requires less than one microsecond regardless of the number of processors. These comparative experiments are enough to prove that the SMCI/Dragon model is efficient and its computational cost is extremely cheap.

5 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 Dragon 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 cache block replacements.

Using the SMCI/Dragon model, several comparative experiments were done which resulted in only 5.4% difference from Archibald's simulation results. In the meantime, each experiment required only the order of micro second to complete it on a desk-top workstation.

Thus we succeeded to apply the original SMCI model to the Dragon protocol with keeping its powerful analytic ability as well as its inexpensive computation cost.

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 our future work.

References

1. Archibald, J., Baer, J.: Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Trans. on Comp. Sys.* **4**(4) (1986) 273-298
2. Dubois, M., Thakkar, S. (ed.): *SCALABLE SHARED MEMORY MULTIPROCESSORS*. Kluwer Academic, (1992)
3. Joe, K., Fukuda A.: An analytic model for a hierarchical parallel system. *Int'l Workshop on Massive Parallelism: Hardware, Software and Applications* (1994) 287-304
4. Joe, K., Fukuda, A.: Analytic modeling of cache coherence based parallel computers. *IEICE Trans. on Inf. and Sys.* (to appear)
5. Mudge, T., Al-Sadoun, H.: A semi-markov model for the performance of multiple-bus systems. *IEEE Trans. on Comp.* **C-34**(10) (1985) 934-942
6. Wang, J., Dubois, M.: Performance comparison of cache coherence protocols based on the access burst model. *Comp. Sys. Sci. and Eng.*, **5**(3) (1990) 147-158