Analysis of Directory Based Cache Coherence Schemes with Multistage Networks

Ashwini K. Nanda
Computer Science Department
Texas A &M University

Hong Jiang
Computer Science and Engineering
University of Nebraska, Lincoln

Abstract

Designing efficient cache coherence schemes for shared memory multiprocessors has attracted much attention of the researchers in the area. *Snoopy* cache protocols have been designed for bus based multiprocessors. However, the snoopy protocols are not applicable to general interconnection networks. On the other hand, the *directory* based cache protocols adapt very well to any kind of interconnection network such as a Multistage Network. Since different protocols have different cost overheads, and may give different performance, the protocol to be used must be wisely selected. Although there has been some simulation studies on the behavior of different directory schemes proposed in the literature, there has been no systematic analytical model for these schemes. In this paper we develop a detailed analytical model of the various directory schemes on a Multistage Interconnection Network. The shared miss ratios are computed analytically, and the performance of the various schemes is compared. Results are presented to show that the directories *do not* form a system bottleneck contrary to popular belief.

1 Introduction

Shared memory multiprocessors with private caches have an inherent cache coherence problem associated with them. The coherence problem arises when multiple copies of a shared memory block are allowed to exist in the local caches and one or more processors are permitted to write on the cached blocks locally. In order to avoid the use of stale data, there must be mechanisms to inform the concerned caches and the memory when a processor modifies a shared block in its local cache. These mechanisms constitute a *cache coherence protocol* for a multiprocessor. The various cache coherence protocols proposed in the literature fall into two principal categories, the *snoopy protocols*[3, 8, 12, 16] and the *directory based protocols*[2, 6, 19, 25].

The snoopy protocols are based on a single bus design. Each cache connected to the multiprocessor bus monitors(snoops) every transaction on the bus and takes appropriate actions as dictated by the protocols. Although the snoopy protocols have been proven to be very efficient for bus based multiprocessors, they are not applicable to multiprocessors based on a general interconnection network that does not have a shared bus.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1992 ACM 089791-472-4/92/0002/0485 $1.50

In contrast, the directory protocols are versatile in nature and can work on any interconnection network. In addition to the state information in the local caches, the directory schemes also store some state informations in *global directories* associated with the memory modules. The various directory schemes differ in the type and amount of state information stored in the global directories and in the manner they handle a particular coherence action. A request that is not satisfied in the local cache is submitted to the corresponding directory. The directory takes the necessary coherence actions and supplies correct data to the requester. It is due to this centralized nature of the cache coherence mechanism in the directories that these schemes do not depend on a particular type of network. Multiprocessor systems based on Multistage Interconnection Networks(MINs) are becoming popular[4, 9, 10, 18] due to the high scalability and cost effectiveness of the MINs. Directory based cache coherence protocols will be naturally suitable for such a multiprocessor. However, there exists ample criticism on the directory schemes with a guess that the centralized directories might become a system bottleneck. The authors are not aware of any implementation of the directory schemes or any comprehensive study, either confirming or countering the above facts.

An evaluation methodology is a necessary tool to determine which of the available options is best suited for a system of given size and characteristics. Analytical models of a system provide a more economical and effective means of evaluation compared to trace driven and event driven simulation models. Considerable amount of effort has been made in the past in developing analytical models for the bus based snoopy protocols [5, 11, 20, 21, 23, 24]. Even though the directory schemes represent an important class of cache protocols, an analytical model for studying their behavior does not exist to our knowledge. A trace driven simulation model was used in[1] to compare the proposed directory methods. This study was limited to a four processor system due to the lack of trace data for larger systems, and the study did not consider the overall system performance.

In this paper we develop an analytical model for evaluating the performance of the directory based cache protocols for a system using a Multistage Interconnection Network. The model uses a markov state approach[24] to calculate the miss ratios for the shared accesses in various protocols. The P-K mean value formula[13] is used to calculate the delays seen by a request at various service centers. The overheads of the directory schemes differ mostly in the number of invalidation and flush signals generated by the directories. Since more than one invalidation signals are generated in a bulk by a single memory request, and the directory can not serve other requests until it has finished sending the invalidations, it is difficult to mathematically model the system exactly. We handle this situation by considering a fictitious source, which generates these signals and presents to the directory, instead of the directory generating them. These signals are given higher priority over the other requests by the directory. The model is used to measure the relative performance

of the different directory schemes with a wide range of system and workload parameters. The analytical model is verified through event driven simulations. Although it is traditionally suspected that the directories may become a system bottleneck, our analysis points to the contrary. Directory utilization is rather low(around 10%) even for a system having as many as 1024 processors.

The rest of the paper is organized as follows. Section 2 discusses the operation of the directory protocols and shows how to derive the shared miss ratios analytically. Section 3 presents the outline of the analytical model. Comparative performance results are presented in section 4, and section 5 concludes the paper.

2 Operation of the Directory Schemes and Calculation of Miss Ratios

2.1 Operation of various schemes

A comprehensive description of all the directory schemes[2, 6, 19] is given in[1]. Here, we present a concise description of the operations of the various schemes. All these directory schemes maintain the state information on shared blocks in each local cache. All of them allow multiple copies of a shared block to exist in the various local caches. There can be as many copies in a system at any time as the number of caches. However, when one of the caches needs to modify its local copy, it has to inform the directory first. The directory invalidates the existing copies in other caches and the block becomes dirty in the requesting cache. Thus, a block can be in one of the following states in a local cache : (1)*invalid*, meaning the cache does not have a valid copy of the block, (2)*valid*, meaning this cache, and possibly other caches have a valid copy, and (3)*dirty*, meaning this cache possesses the only copy in the system, and it has been modified. In addition to the above information in the local caches, these schemes also keep some state information in the *directories* associated with the memory modules. In Tang's scheme[19], the state information of each of the local caches is duplicated in the directory. Censier and Feautrier's scheme[6] keeps the same state information, but in a tabular fashion. Operation of these two schemes are the same.

Archibald and Baer's scheme[2] does not keep the state information of the individual caches in the directory; it keeps the consolidated state information of all caches by using only two bits per shared block. The informations kept in the directories are : (1)$INVALID$, meaning the block is not present in any local cache, (2)$VALID$, meaning the block is *valid* in an unspecified number of local caches, (3)$VALID1$, meaning the block is *valid* in exactly one cache, (4)$DIRTY$, meaning the block is *dirty* in some cache. The directories in this scheme do not know the identity of the local cache having a *valid* or a *dirty* copy. Since this scheme keeps less state information in the directory, it has a penalty of more coherence overheads.

Agarwal et.al.[1] developed a generic terminology to describe the above protocols in a systematic manner. They also suggested a new directory protocol(called dir_1NB), which does not allow a shared block to exist in more than one cache at a time. The directory entry for a block in this scheme keeps a pointer to the cache that contains the block. If a cache has a block in the *valid* state or in the *dirty* state, and another cache requests a copy of the block, the present owner invalidates its copy before the the copy is supplied to the requester. This is the simplest of the directory schemes and the details of this protocol are similar to the other protocols as given in[1].

In this paper we adopt a simpler terminology to denote a particular directory scheme. Since there is no broadcasting medium in a MIN, we drop the B/NB option of[1]. In general we will call a scheme as dir_x, where x is the number of processor indices kept in the directory. Archibald and Baer's scheme[2] will be denoted as dir_0, since it does not keep the information on any particular cache in the directory. Agarwal et.al.'s single copy scheme[1] will be denoted as dir_1. Tang's[19], as well as Censier & Feautrier's[6] schemes will be denoted as dir_N, since they keep the information on all the local caches in the directory. Although the dir_0 scheme does not keep any information on a particular cache, it allows all the caches to have a valid copy of a block.

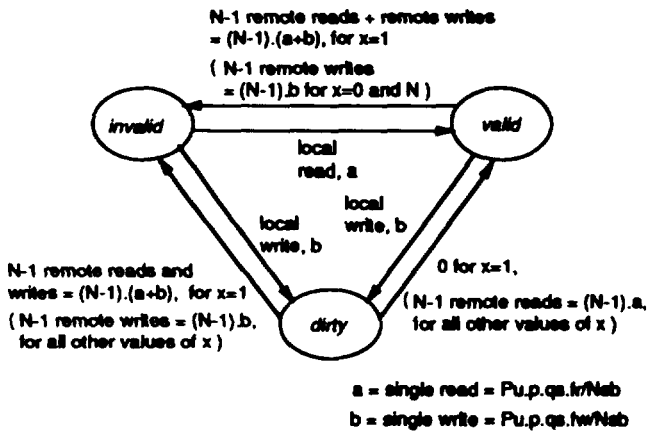
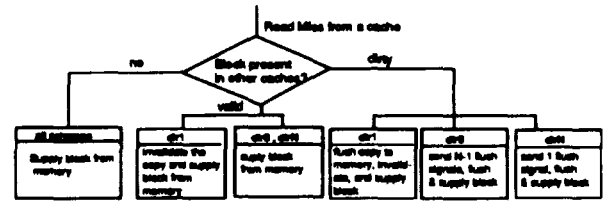
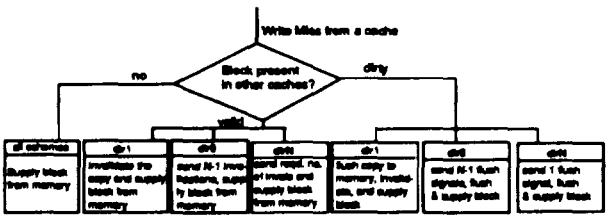


Figure.1 State Transition Diagram of a Shared Block in a Local Cache for dirx scheme

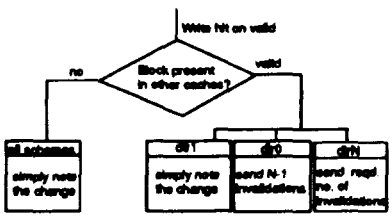
The above dir_x schemes basically operate in the same way except for some minor differences. Figure.1 shows the state diagram of a shared block in a local cache for the dir_x schemes. This diagram indicates when the state of a block must change in the local cache. The coherence actions taken by the directories in the dir_0, dir_1 and dir_N schemes in the events of a read miss, a write miss, and a write hit on a valid copy are depicted in figures 2.a, 2.b and 2.c respectively. On a read miss, the request goes to the directory. If the block is *dirty* in another cache, then the block is flushed(retrieved) from the owner cache, written into the memory, and supplied to the requester as a *valid* copy. Otherwise, the block is supplied from the memory. The dir_1 scheme, in addition, invalidates any valid copy in the system. On a write miss, the directory invalidates the other valid copies, if any, and supplies the block from the memory. If the block was *dirty* in some cache, it is flushed to the memory first and then invalidated. For a write hit on a dirty, copy no coherence action is needed. On a write hit on a valid block, the requester sends an invalidation request to the directory and the directory invalidates the other existing valid copies, if any. This chart shows the basic differences in the number of signals generated by the directories in the three schemes. Since the dir_N and dir_1 schemes know the identity of the caches having a valid or dirty copy, they generate the exact numbers of invalidation and flush signals as required. However, the dir_0 scheme does not know the location of the valid or dirty copies, and when required, it has to send the invalidation and flush signals to all caches except for the requester. The dir_1 scheme does not allow more than one cache to keep a copy of a block which results in more misses in the local caches than in case of the dir_0 and dir_N schemes. These differences in overheads significantly affect the overall performance of the schemes as will be shown in section 4. In the next subsection we describe the system and workload assumptions used



(a) read miss from a cache



(b) write miss from a cache



(c) write hit on valid from cache

Figure.2 Actions taken by the directories of various schemes on different events

in our study.

2.2 System and Workload Models

The system configuration used for the various schemes is shown in figure.3. There are N processors in the system, each having a local cache. The processors are connected to N memory modules using a MIN. Each memory module has a directory associated with it for keeping state informations and taking cache coherence actions. A delta network[17] comprising of a forward and a backward network is used for the MIN. The system is assumed to be synchronous and packet switched. The MIN switch service time forms the basis of the system cycle time. In further discussions, *cycle* will mean this system cycle. The service times of the caches, the directories and the memories are integral multiples of this cycle time. The processors are represented as delay centers, and in a given cycle, they submit memory requests to their caches with some given probability. However, a processor waits until its previous memory request is satisfied, before submitting another request. The system parameters are defined below.

N : no. of processors in the system

$k * k$: size of the MIN switches

t_d : directory access time

t_i : directory controller service time for generating a single invalidation signal

t_c: cache cycle time

t_s: MIN switch service time

t_m: memory service time

p: probability that a processor submits a memory request in a given cycle provided it is busy

P_u: processor utilization

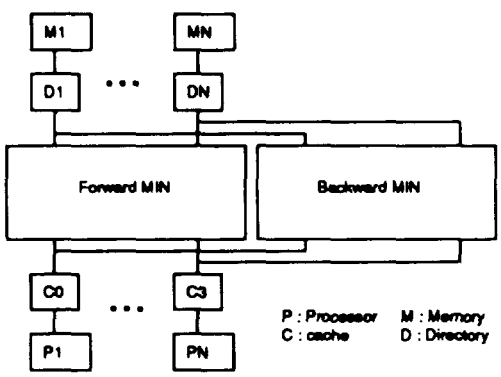


Figure.3 Multiprocessor system using directory schemes

Workload Parameters

In the absence of measurement data on the memory reference patterns for large multiprocessors, one has to resort to some kind of synthetic patterns while analyzing such systems. The workload model used for this analysis is an extension of the workload developed in [7]. The memory reference string of a processor consists of two streams, one for private and read-only shared blocks and the other for the read-write shared blocks. For convenience, we will call them *private requests* and *shared requests* respectively . A given memory request is to a shared block with probability q_s and to a private block with probability 1-q_s. When $q_s = 0$, there are no accesses to shared blocks and hence there is no overhead of cache coherence. As q_s increases the cache coherence overhead also increases and the performance is affected more. A private request is a hit in the local cache with probability h and a miss with probability $1 - h$. The number of read-write shared blocks (or simply shared blocks) in a system, N_{sb}, is assumed to be fixed. A memory request is a write with probability f_w and a read with probability $f_r = 1 - f_w$. In invalidation based cache coherence protocols, the parameter f_w plays a major role. When $f_w = 0$, there are no writes and therefore there is no invalidation traffic. As f_w increases, the number of write requests increases and the invalidation traffic also increases causing performance degradation of the system. The various workload parameters are summarized below.

N_{sb}: no. of shared blocks in the system

q_s: prob. that a request is to a shared block (prob for a private block is $1 - q_s$)

f_w: prob. that a request is write ($f_r = 1 - f_w$)

h: prob. that a private request is a hit in the cache

2.3 Calculation of Shared Miss Ratios

The miss ratio of the shared block references is the same as the probability that the block is *invalid* in the local cache. There are two options available regarding the state probabilities. First, we can choose some arbitrary values for the local state probabilities independent of the workload and system parameters and proceed with the analysis. Second, we can derive the local state probabilities from the markov state diagram of the protocol as functions of the system and workload parameters as in [24] and then use them in the analysis. Since the state probabilities are highly dependent on workload parameters in practice, we will follow the second approach in our study. We will assume infinite local caches in order to avoid block replacements. However, the effect of replacements can be easily incorporated in the state diagrams by appropriately modifying the transition rates. The markov

state diagram of a block in a local cache in the three directory schemes under consideration was shown in figure.1. Since the dir_0 scheme allows all the caches to have valid copies, the transition rates for this scheme are the same as that of the dir_N scheme. The transition rates for the dir_1 scheme are different from the others as they allow only one copy in the system at any time. The various state probabilities are defined below.

p_i : probability that a block is in *invalid* state in the local cache

p_v : probability that a block is in *valid* state in the local cache

p_d : probability that a block is in *dirty* state in the local cache

By solving the flow balance equations of the state diagram, we get the following expressions for the above state probabilities.

For dir_0 and dir_N :

$$p_d = \frac{f_w}{(N-1).f_r + N.f_w}$$

$$p_v = \frac{f_r.(1 + (N-2).p_d)}{N.f_w + f_r}$$

$$p_i = 1 - p_d - p_v$$

For dir_1 :

$$p_i = \frac{N-1}{N}$$

$$p_v = \frac{f_r.p_i}{N.f_w + (N-1).f_r}$$

$$p_d = 1 - p_i - p_v$$

The shared miss ratio p_i for the different schemes are shown in figures 4 and 5 as a function of the write ratio f_w and the system size N respectively. The miss ratio for the dir_1 scheme does not vary with the write frequency since the protocol allows only a single cache to have a copy at any time. On both a read request and a write request from another cache this copy is invalidated. Thus, the write frequency does not have any special effect on the miss ratio. The miss ratio in case of the dir_0 and dir_N schemes is low for small values of write frequencies. When write frequency increases, there are more invalidations, and there are more misses. When the system size increases, with a constant write frequency, the miss ratios for all the schemes increase. This increase in miss ratio occurs due to the fact that the frequency of invalidations increases when the number of processors, potentially sharing the block, increases. The effect is more pronounced in case of the dir_0 and dir_N protocols. However, as seen from both figures 4 and 5, the dir_1 scheme always has a higher miss ratio than the other two schemes. The miss ratio along with the frequency of write hits on *valid* blocks determines the amount of shared memory traffic released by a local cache to the network and the directory. The differences in this traffic along with the number of invalidation and flush signals generated by the directories give rise to the difference in performance of the various schemes. It may be pointed out that although the dir_1 scheme generates the highest miss ratio among the three protocols, the overall performance will not be that bad because the dir_1 scheme needs less invalidations. We will see these results in section 4.

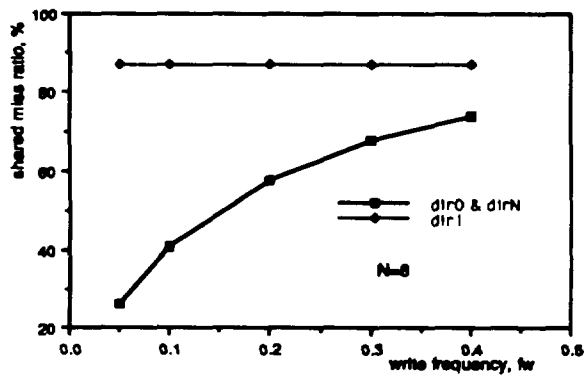


Figure.4 shared miss ratio as a function of the write frequency

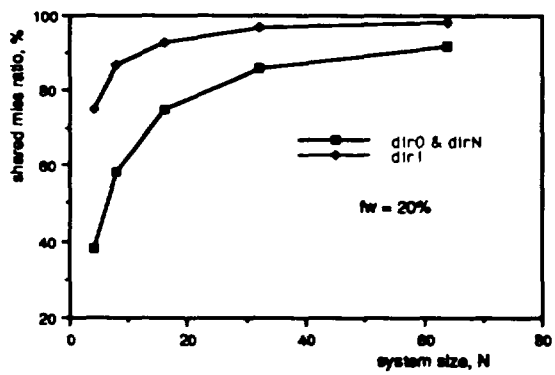


Figure.5 shared miss ratio as a function of the system size

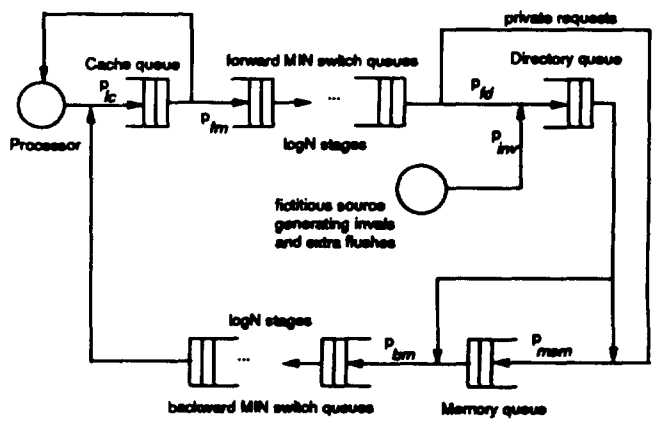


Figure.6 Queueing Models of Various Service Centers

3 Outline of the Analysis

A memory request generated by a processor will visit different service centers determined by the type of the request and the state of the block in the local cache. The delay seen by a request at any center, or the *response time* at that center will depend on the service demand and the amount of input traffic at that center. The various service demands or service times are assumed to be fixed and have been discussed along with the system parameters. The traffic at an input of a service center can be represented by the probability that there is a request coming into that input in a given cycle, or in short, the *request probability*. The request probabilities at different service centers, are in general, functions of the processor utilization, the workload parameters and the probabilities of a block being in different states in the system. The global state probabilities depend on the local state probabilities and the local state probabilities in turn depend on the workload parameters and system size. The local state probabilities were defined and derived in the previous section. We define below various conditional global probabilities to be used in the analysis.

$p_{(V/i)}$: probability that there are *valid* copies in the system, given that the block is *invalid* in the local cache

$p_{(V/v)}$: probability that there are other *valid* copies in the system given that the block is *valid* in the local cache.

$p_{(D/i)}$: probability that there is a *dirty* copy in the system, given that the block is *invalid* in the local cache

The conditional global probabilities will depend on the local state probabilities and can be expressed as follows :

$$p_{(V/i)} = (N-1).p_v.(1-p_v)^{N-2} \quad for\ x = 1,\ and$$

$$p_{(V/i)} = 1-(1-p_v)^{N-1} \quad for\ x = 0\ and\ x = N.$$

$$p_{(V/v)} = 0 \quad for\ x = 1,\ and$$

$$p_{(V/v)} = \sum_{j=1}^{N-1} C_j^{N-1}.p_v^j.(1-p_v)^{N-1-j} \quad for\ x = 0\ and\ x = N.$$

$$p_{(D/i)} = (N-1).p_d \quad for\ all\ x.$$

A Queueing Network Model

The closed queueing model of the system is shown in fig.6. Since we assume a homogeneous system, all the service centers in a given stage, e.g. all the local caches or all the directories, can be represented by the behavior of any single center in that stage. We will assume infinite buffer lengths for holding requests in the queue at the input of every service center. A processor first submits its requests to the cache. These requests are either satisfied by the cache or are sent to the forward network. The forward network transmits all the shared requests to the directory and all the private requests to the memory. The directory also receives the requests from a fictitious source for generating invalidations. The rationale behind this source will be explained in a short while. The directory passes the invalidation and flush signals to the backward MIN and the unsatisfied shared requests and writebacks to the memory. The memory sends all its replies to the backward network. The backward network passes all the replies and control signals to the cache. A request will visit some combination of the above service centers depending on the type of the request and the state of the block being requested in the system.

The reason for including a fictitious source at the input of a directory queue is as follows. A local cache sends only a single invalidation request to the directory. The directory then generates a number of invalidation signals depending on the directory scheme. These generated invalidation signals do not represent any real request from a processor. But the directory has to spend service time on them and it can not serve any other request until all the invalidation signals have been generated. We represent this phenomenon mathematically by imagining a fictitious source which generates the invalidation signals and presents to the directory for service. The directory gives priority to these signals over all other requests such that as long as there is a request from this source, other requests will not be served. The invalidation signals are generated in a burst when there is an invalidation request from a processor. We make a simplifying assumption that these signals are uniformly distributed in time and the probability that there is a request from this source at any cycle is time-independent. Though this assumption is inaccurate, it makes the analysis simpler and does not degrade the overall accuracy of the analysis significantly. The fictitious source also includes the generation of the extra flush signals required in the dir_0 scheme.

Request Probabilities

The *request probability* at an input of a service center was defined earlier as the probability that there is a request at that input in any given cycle. The different request probabilities shown at the inputs of these queueing centers in figure.6 are defined below.

p_{lc} : probability that there is a request at the input of a local cache

p_{fm} : prob. that there is a request at an input of a forward MIN switch

p_{bm} : prob. that there is a request at an input of a backward MIN switch

p_{fd} : prob. that there is a request from the forward MIN to a directory

p_{mem} : prob. that there is a request at an input of a memory module

p_{inv} : prob. that there is an invalidation (or flush) request at an input of a directory

The above request probabilities are functions of the processor utilization, the probability that a memory request is generated, the private and shared reference ratios, the read write frequencies, private and shared miss ratios, the local and global state probabilities of the block and the system size. Since the three coherence protocols are essentially the same except for the number of invalidation and flush signals generated, the expressions for the request probabilities except for p_{inv} will be the same for all. We will derive the expression for p_{inv} below. The expressions for the other request probabilities are derived in [15].

The invalidation signals are generated by the directory when (a)there is a write miss and the block is *valid* in another cache, (b)there is a write hit on a *valid* block and there are other *valid* copies in the system, and (c)when there is a read request for a block that is already *valid* in x other caches. Let us define n_1, n_2 and n_3 as the average number of copies to be invalidated in case a, case b and case c respectively. Then, $n_1 = (N-1).p_v^j.(1-p_v)^{N-2}$ for $x = 1$, $n_1 = (N-1).p_v$ for $x = N$ and $n_1 = (N-1)$ for $x = 0$. $n_2 = 0$ for $x = 1$, $n_2 = (N-1).p_v$ for $x = N$ and $n_2 = N-1$ for $x = 0$. Note that case c does not arise for $x = 0$ or $x = N$ and the number of invalidation signals generated is always 1 for

$x = 1$. Therefore, $n_3 = p_{V/i}$ for $x = 1$ and $n_3 = 0$ for $x = 0$ and $x = N$.

The flush signals are generated by the directory in case of a shared read miss as well as a shared write miss on a block that is *dirty* in another cache. Let us define n_4 as the number of *unnecessary* flush signals generated for each such request. Then $n_4 = N - 2$ for $x = 0$ and $n_4 = 0$ otherwise. Hence by our definition of p_{inv} we have

$$p_{inv} = P_u.p.q_s.(n_1.f_w.p_i.p_{(V/i)} + n_2.f_w.p_v.p_{(V/v)} + n_3.f_r.p_i.p_{(V/i)} + n_4.p_i.p_{(D/i)}) \quad for\ all\ x.$$

Response Times

We will define the *response time* of a service center as the total delay, i.e. the queueing delay plus the service time, faced by a request at the particular service center. In order to make the analysis simpler, each service center in the system is considered in isolation from the other centers. The response times of various service centers are enumerated below.

r_c : response time of a cache controller
r_f : response time of a forward MIN switch
r_b : response time of a backward MIN switch
r_d : response time of a directory controller
r_m : response time of a memory module

The response time at an individual service center will depend on the number of inputs to that center, the request probability at that center and the service demand of a request at that center. The well known Pollaczek-Khinchine(P-K) mean value formula[13] will be used to derive these response times. The details of the derivations are given in [15]. The expression for the directory controller response time is relatively more complex than others due to the requests from the fictitious source which must get priority. For this queue, first the utilization due to the high priority requests is derived, and then the response time is modified according to the fraction of directory time available for the other requests.

Overall Delays and Processor Utilization

At any point of time a processor is either busy doing some internal computations or is waiting for the response to a memory request. The *Processor Utilization*, P_u is defined as the fraction of time a processor is busy doing some useful computation. If it is busy, then it submits a memory request to its cache in a cycle with probability p. In case of a private request, the block is immediately supplied if it is a hit, and if it is a miss, then the block is supplied by the memory through the MIN with the same amount of delay for both a read and a write. In case of a shared request, the required consistency action is taken by the cache controller and the corresponding directory near the memory. The request is satisfied with a variable amount of delay depending on the state of the block and whether the request is a read or a write. The processor utilization can be expressed in terms of these *overall delays*. If we define

d_p : overall delay for private requests
d_r : overall delay for read requests to shared blocks
d_w : overall delay for write requests to shared blocks,

then P_u will be given by,

$$P_u = \frac{1}{1 + p.((1 - q_s).d_p + q_s.f_r.d_r + q_s.f_w.d_w)}$$

The values of d_p, d_r and d_w will be the aggregated delays across all the centers visited by the corresponding request.

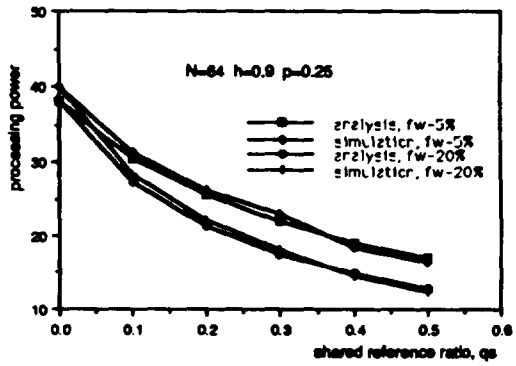


Figure.7 Comparison of analytical and simulation results for dirN scheme

These delays will be functions of the service center response times, the miss ratios and the local and global state probabilities and are derived in [15].

The *system power* is defined as the sum of the processing powers of all processors, or $N.P_u$, and will be used as the performance measure in our study. The delays d_p, d_r and d_w will depend on the amount of traffic in the MIN, which in turn is a function of P_u itself. Thus we get a nonlinear equation with P_u as the single variable, which can be solved by using iteration techniques as discussed at the end of this section. More details of the analysis can be found in[15].

Numerical Solutions and Validation of Results

Having found the relation between P_u and the various delays, we can employ any standard iterative technique to solve the equation for P_u for any directory scheme. A higher order iteration would give faster convergence. Usually the speed of convergence also depends on the initial value. With some experimentation, $P_u = 1/N$ was found to be a good initial value for this problem and usually the convergence took 2 to 6 iterations. Due to finite precision of the computers, when a number is subtracted from another comparable number, the percentage of error may become too high because of truncation. In such a case the iteration tends to oscillate. This can happen when the utilization of any service center comes very close to 1 (saturation). When this happens, the iteration process can be stopped and the value of P_u that caused this saturation can be calculated from the equations for that center. The analytical solution takes only of the order of a second to run on a sparcstation/3. An event driven simulation model was developed[15] to verify the analytical model presented in this paper. The analytical results match with the simulation results with an accuracy of 5 to 6%. A sample comparison of the analytical and simulation results is presented in figure.7. The curves are drawn for a dir_N scheme on a system with 64 processors, with different values of the shared reference ratio and write frequency.

4 Results and Comparison

The choice of system parameters, especially the service demands at various centers has a significant effect on the performance of a protocol on a given configuration. The analysis developed in section 3 can be applied to any set of parameters. In this section we use some parameters values, that we believe are typical, to compare the performance of the three directory schemes discussed in section 2. We choose the cache service time(t_c) to be 1 cycle, since the caches are made of high speed memories. The main memories are normally a few times slower than the caches, and therefore we

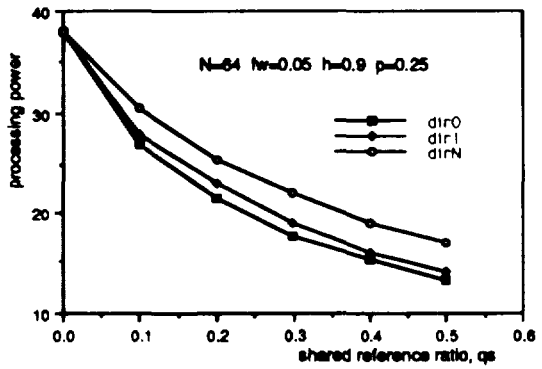


Figure.8 comparison of dirx schemes, varying shared reference ratio

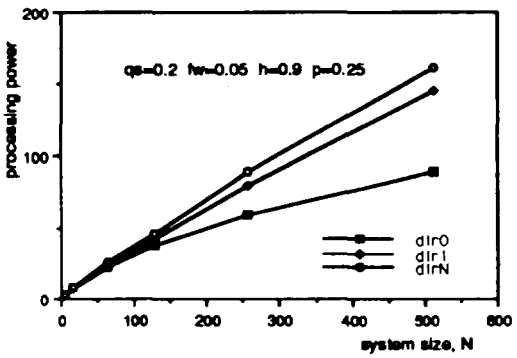


Figure.9 comparison of dirx schemes, varying system size

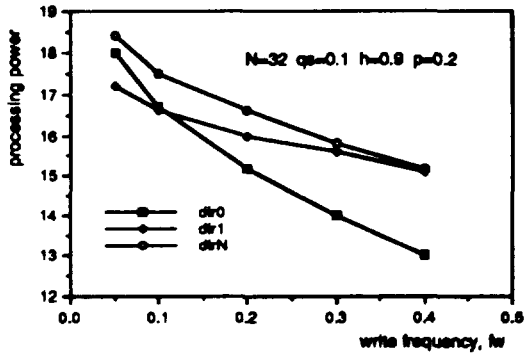


Figure.10 comparison of dirx schemes, varying fw

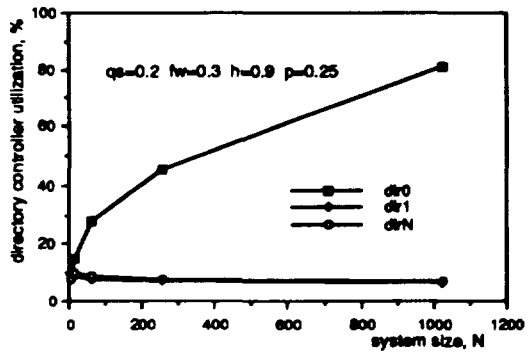


Figure.11 utilization of the directory controller as system size grows

chose memory service time(t_m) as 4 cycles. When the directory is designed as a part of the memory module, it will have an identical access time as the memory(t_d=4 cycles). However the generation of invalidation signals can be done at a faster speed after the directory is accessed to find out the status of the block in various caches. We assume that each such invalidation or flush signal generation takes one cycle(t_i=1 cycle). The MIN switch size is chosen to be 2*2 for the results in this section. Although we chose the above parameters as specific instances, their exact values will depend on the system being designed. We will use the processing power(sum of processor utilizations) of the system as the performance measure in the following discussions.

Figure.8 shows the relative performance of the three directory schemes for a system with 64 processors. For $q_s = 0$, all the memory requests are to the private blocks, and all the schemes have identical performance. By increasing q_s, the coherence overheads increase, which leads to a performance degradation in all the schemes. The dir_1 scheme performs slightly better than the dir_0 scheme in spite of having a higher miss ratio. The reason is the large number of invalidation and flush signals generated by the later. The dir_N scheme performs better than the dir_1 scheme even though it incurs more invalidation overheads than the later. This superiority in performance can be attributed to the lower miss ratio of the dir_N scheme over the dir_1 scheme for the chosen value of f_ω. Figure.9 shows the effect of increasing the system size on performance. The dir_N scheme is always superior to the other two for the chosen values of workload parameters. It can be noticed that the performance of the dir_0 scheme is the one that is affected the most when the system becomes larger. The reason is that the overheads of this scheme grow directly with N (always N-1 invalidations, irrespective of how many are really needed). The dir_0 scheme is well suited to systems where broadcasting is allowed. If a MIN can be designed with broadcasting capability, then the dir_0 scheme will perform well.

It was observed in section 2 that the miss ratio of the dir_0 and dir_N schemes increases with write frequency while the miss ratio of the dir_1 scheme remains constant. The invalidation overheads also increase with an increase in write frequency as they are directly related. The above facts suggest that although the dir_N scheme may perform better than dir_1 for lower values of f_ω, its performance will degrade at a faster rate than the later with increase in f_ω. Figure.10 shows that this is in fact the case. Initially, dir_N is much superior in performance, but after f_ω is around 0.30, the dir_1 scheme performs nearly as good as the dir_N scheme with the chosen parameters. For lower values of f_ω, the dir_0 scheme also performs better than the dir_1 scheme. For higher values of f_ω, dir_1 performs better. An additional advantage for the dir_1 scheme is that the storage requirements for this scheme is only slightly higher than the dir_0 scheme, but substantially less than the dir_N scheme. Therefore, the cost/performance ratio of the dir_1 scheme may prove to be more attractive.

Since the directory controllers handle all the coherence actions, one might suspect that they may soon become a bottleneck. We show the directory utilization of the three schemes as a function of the system size, for relatively higher q_s and f_ω, in figure.11. While the directory in the dir_0 scheme starts saturating after N=256, the directory utilization of the dir_N scheme remains low even up to $N = 1024$, at around 10%. The dir_1 scheme has similar low directory utilizations. The directory schemes adapt well to the growth in system size, simply because with the growth in the number of memory

modules, the number of directories also increases and the requests get distributed.

5 Conclusion

The directory based cache protocols provide an important alternative in designing large scale cache coherent multiprocessors. In this paper we presented an analytical model for evaluating the performance of various directory schemes on a MIN based multiprocessor. The miss ratios for the shared memory references for different schemes were derived analytically. A model was devised to handle the bulk generation of the invalidation and flush signals mathematically. Overall analysis of the schemes showed that the dir_1 scheme performs nearly as good as the dir_N scheme. Although dir_N performs better in most cases, the saving of directory storage space in the dir_1 may make the later scheme more attractive. Performance of the dir_0 scheme is severely affected by the increase in system size due to the generation of a large number of invalidation signals. The directories in the dir_0 scheme also become saturated fast. However, the directories in the dir_1 and dir_N schemes remain rather underutilized even for a system containing as many as 1024 processors. We would like to point out here that since there is not a single implementation of a directory scheme reported in the literature, we had to second guess many operations. Hence the results presented in this paper should not be over emphasized. The analytical technique presented here, however, will be helpful for evaluation and quick prediction of performance of a directory based cache coherent multiprocessor when implemented.

Acknowledgement

The authors wish to extend their deep gratitude to Dr. L.N. Bhuyan for his support for this research and his invaluable comments on this paper. This research is supported by the NSF grant MIP-9002353.

References

[1] A.Agarwal, R.Simoni, J.Hennessy and M.Horowitz, "An evaluation of directory schemes for cache coherence," *The 15th Ann. Intl. Symp. On Comp. Arch.*, pp. 280-289, June 1988.

[2] J.Archibald and J.L.Baer, "An economical solution to the cache coherence problem," *The 11th Ann. Intl. Symp. On Comp. Arch.*, pp. 355-362, June 1984.

[3] J.Archibald and J.L.Baer, "Cache coherence protocols : evaluation using multiprocessor simulation model," *ACM Transactions on Comp. Systems*, pp.273-298, Nov 1986.

[4] BBN Laboratories Inc., "Butterfly Parallel Processor Overview," Dec. 19, 1985.

[5] L.N.Bhuyan,B.C.Liu and A.Irshad, "Analysis of MIN based multiprocessors with private cache memories," *Proc. of the 1989 Intl. Conf. on Parallel Processing*, 1989.

[6] L.M.Censier and P.Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, pp. 1112-1118, Dec 1978.

[7] M.Dubois and F.A.Briggs, "Effect of cache coherency in Multiprocessors," *IEEE Transactions on Computers*, pp. 1083-1099, Nov 1982.

[8] J.R.Goodman, "Using cache memory to reduce processor-memory traffic," *The 10th Ann. Intl. Symp. On Comp. Arch.*, pp. 124-132, June 1983.

[9] D.D.Gajski et. al. "Cedar - A Large Scale Multiprocessor" *Proc. of the 1983 Intl. Conf. on Parallel Processing*, pp. 524-529.

[10] A.Gotlieb et. al., "The NYU Ultracomputer - Designing an MIMD shared memory parallel processor," *IEEE Transactions on Computers*, pp. 175-189, Feb. 1983.

[11] A.G.Greenberg and I.Mitrani, "Analysis of snooping caches," *The 12th Intl. Symp. On Comp. Performance* , Dec 1987.

[12] R.Katz, S.Eggers et. al. "Implementing a cache consistency protocol" *The 12th Ann. Intl. Symp. On Comp. Arch*, pp. 276-283, June 1985.

[13] L.Kleinrock, *Queueing Systems Vol.1, Theory*, New York Wiley-Interscience 1975.

[14] C.P. Kruskal and M. Snir, "The Performance Of Multistage Interconnection Networks for Multiprocessors," *IEEE Transactions on Computers*, pp. 1091-1098, Dec. 1983.

[15] A.K.Nanda, "Cache Coherent Architectures for Large Scale Multiprocessors", Ph.D. Thesis under preparation at Texas A&M University.

[16] M.Papamarcos and J.Patel, "A low overhead coherence solution for multiprocessors with private cache memories," *The 11th Ann. Intl. Symp. On Comp. Arch.*, pp. 348-354, June 1984.

[17] J.H.Patel, " Performance of Processor Memory Interconnections for Multiprocessors," *IEEE Transactions on Computers*, pp. 771-780, Oct. 1981.

[18] G.F.Pfister et.al., "The IBM Research Parallel Processor Prototype (RP3) : Introduction and Architecture," *Proc. of the 1985 Intl. Conf. on Parallel Processing*, pp. 764-771.

[19] C.K.Tang,"Cache system design in tightly coupled microprocessor systems," *AFPIS proc., Natl. Comp. Conf.*, pp. 749-753,1976.

[20] M.K.Vernon and M.A.Holliday, "Performance analysis of multiprocessor cache coherency protocols using generalized timed petrinets," *Proc. ACM SIGMETRICS Conf.*, pp. 9-17,1986.

[21] M.K.Vernon et.al., "An Accurate and Efficient Performance analysis for Multiprocessor Snooping Cache-Consistency Protocols," *The 15th Ann. Intl. Symp. On Comp. Arch.*, pp. 308-315, June 1988.

[22] A.W.Wilson, "Hierarchical cache/bus architecture for shared memory multiprocessors," *The 14th Ann. Intl. Symp. On Comp. Arch.*,pp. 244-252, June 1987.

[23] Q.Yang and L.N.Bhuyan, "A queueing network model for a cache coherence protocol on multiple bus multiprocessors," *Proc. of the 1988 Intl. Conf. on Parallel Processing*, pp. 130-137, Aug.1988.

[24] Q.Yang, L.N.Bhuyan and B.C.Liu, "Analysis and comparison of cache coherence protocols for a packet switched multiprocessor," *IEEE Transactions on Computers*, pp. 1143-1153, Aug. 1989.

[25] W.C.Yen et.al., "Data coherence problem in a multicache system," *IEEE Transactions on Computers*, pp. 56-65, Jan. 1985.