Single Copy vs. Multiple Copies Cache Coherence Protocols for Hierarchical Bus Multiprocessors

Jason Tsui and Mokhtar Aboelaze
Department of Computed Science
York University
N. York, Ontario M3J 1P3 CANADA

Abstract

Reducing memory access time is a very important factor in increasing the performance of multiprocessors. This could be achieved by using caches to hide the memory latency. In multiprocessors, caches introduce another problem, namely, the cache coherence problem. *In this paper, we investigate single copy cache coherence protocols for multiprocessors with a hierarchical bus structure. We introduce two different single copy cache coherence protocols and a third protocol that allows a limited number of copies in the system. We compare our protocol to some of the well known cache coherence multiple copies protocols for hierarchical systems using a synthetic workload model.*

1 Introduction

With the switching speed, and consequently the processor speed, approaching its physical limit, parallel processing is considered the only approach to achieve the high performance computing power required for the ever increasing requirements in scientific and non scientific applications. One of the major obstacles in designing high performance computers is the uneven growth of the processor speed and memory speed. This problem is usually addressed by using *caches*. A cache is a small but fast memory that holds the most frequently accessed data. The cache speed is the same as the processor speed, thus could be accessed in one processor cycle. Adding a cache to a uniprocessor system involves the added complexity of managing the cache and ensuring that it holds the data that is most probably to be referenced in the near future.

In multiprocessors, there is an added problem, namely, the cache coherence problem. In multiprocessors, each processor has its own cache, data blocks may be cached in more than one cache at the same time. Modifying any data block in one of the caches, leaves the other copies in other caches *stall* i.e. the contents of the cache is outdated. That is why we have to enforce a cache coherence policy to guarantee the consistency of the data.

Cache coherence protocols can be divided into two major categories, software-based protocols, and hardware-based protocols. Software-based protocols depends on the compiler to tag the different data blocks either cachable or non-cachable in order to avoid caching a shared block. Software-based protocols may lead to a poor performance due to its complexity and its conservative policy. On the other hand, although hardware-based protocols require additional hardware, their performance is better than software-based protocols. Hardware-based protocols may be further classified into invalidate-based, or write propagate protocols. In invalidate-based protocols, if a block of data exists in more than one cache, before any processor modifies one of the copies, this processor must send a signal to invalidate all the other copies in the system. Thus, making its copy the only copy in the system, and could be modified without any inconsistencies. In write propagate protocols, a write to a certain block is not complete until the modified data is propagated to all the other copies of the same block in the system. Thus, guaranteeing a consistent view of the memory.

In this paper we propose two single copy cache coherence protocols for hierarchical bus multiprocessors. We also propose another protocol that allows a limited sharing of the blocks between more than one processor if the processors belong to the same cluster, thus combining the advantages of the single copy protocol with the advantages of multiple copies protocols. We compare the performance of our protocol with the previously known protocols for hierarchical bus architecture using a probabilistic workload access model.

The organization of this paper is as follows, in section 2 we briefly review the existing cache coherence protocols. In Section 3 we present the workload model we use in our simulation. In section 4 we present our proposed protocols. In section 5 we present simulation results for our protocol and compare it with other protocols. Section 6 is a conclusion.

2 Cache Coherence Protocols

One of the most simple cache coherence protocols is the *snoopy cache coherence protocol* [Goo83], [EgK89], and [ThiS87]. The snoopy protocol is suitable for a bus-based multiprocessors, where every transaction by each processor is heard by all other processors. In snoopy cache coherence protocols, each processor has a snoopy controller that is continuously monitoring the bus. A block in a processor cache can be in one of two states, either shared (consistent with the main memory, and possibly other copies exist in other processors caches), or exclusive (only copy in the system). A block in an exclusive state could be either clean (not modified and consistent with the main memory) or dirty (modified, must be written back upon replacement). Every snoopy controller monitor every request on the bus. The requests could be one of three categories: read (local or external), write (local or external), or invalidate. Based on the type of request, an action is taken to guarantee the consistency of the data. This action could be either invalidating an existing block, modifying an existing block, supplying a copy of an existing block to another processor cache or changing the state of this block. The major drawback of snoopy protocols is the single bus. The single bus is suitable for a small number of processors. However, with increasing the number of processors in the system, the bus becomes a bottleneck, and the system performance degrades considerably.

Another class of protocols that is successfully used with multiprocessors with a large number of processors is the directory based cache coherence protocols [Ste89], [ThD90], and [ChK91].

The idea behind the directory-based cache coherence protocols is to supply enough information for the different processors to know the locations of all other copies of each block in the system. This is usually implemented by using a directory to hold information about the location of every cached block in the system [ChK91], [AgS88]. This directory is usually kept in the main memory, and every reference to a shared block must go through this directory.

The directory could be implemented as a vector of length N for every cache block, where N is the number of processors (caches) in the system. If processor i contain a copy of this block, then the i^th entry of this vector is set to 1. If a processor wants to modify block j, the cache coherence protocol checks the j^{th} vector, and sends either invalidation, or write update signal to every cache with a copy of block j. The main disadvantage of this technique, is that with increasing the number of processors, the overhead in storing the directory is increased.

Since simulation studies have shown that the probability of a large number of processors sharing a single data block is small [EgK88], [ChK91], We can decrease the amount of storage required by limiting the number of entries (the number of caches allowed to share a single block) to a small number k. If the $k + 1^{st}$ processor requests a copy of the, we have to invalidate one of the existing copies. This method is known as *limited pointer directories.*

Anderson and Baer in [AnB93] proposed a directory based protocol for use with systems with hierarchical buses. Their architecture assume private caches at the processor site, and shared caches on the buses at higher levels. they proposed a directory based cache coherence protocols. In their protocol they used a transitional states in order to deal with the problem of looking for a block that is in motion between two caches.

Recently, there has been some interest in single copy cache coherence protocols. In single copy protocols, only one copy of every block is permitted to exist at any time in the system. Thus, we don't have to worry about cache coherence at all. The disadvantage of these protocols is its inability to share data between two processors, even if the processors are reading it only. In cases where two processors access the same block simultaneously, ping-pong effect arises.

Mizrahi et al in [MiB89] proposed a memory hierarchy network (MHN) and a single copy protocol in which the cache coherence is guaranteed by disallowing multiple copies of the same memory block. They also proposed that the MHN is used for handling shared data, while a separate network is used for handling private data and requests. The MHN idea is to consider the main memory as the root of a tree, while processors and their private caches are considered to be the leaves of that tree. Each internal tree node (excluding the root and the leaves) consists of a cache memory to hold data, a directory to locate the requested data block, and switching mechanism to forward the requested block. The authors also developed a data migration policy to control the movement of the memory blocks in order to minimize the average memory access time.

Omran and Aboelaze in [OmA94] has proposed a single copy cache coherence protocol for multiprocessors with multistage interconnection networks. They proposed a network where the switches has caches, and data can be stored in the switches caches. They investigated both cases where caches are allowed only in the first stage vs. cache allowed in any internal switch.

3 Workload Model

In uniprocessor simulation studies, real address traces have been used in order to measure protocol performance. However, in multiprocessors, the real-trace driven simulations have been less successful. Some researchers have used real parallel program traces to simulate cache coherence protocol in multiprocessors. These traces are usually for a fixed number of processors, and depends on the particular machine being used. It is difficult to measure the effect of changing the number of processors in the system, the percentage of the write access, or the degree of coupling between the different processes. In this paper, we use a flexible workload model that can synthetically emulate the reference pattern of a variety of applications.

Our synthetic trace is generated using four parameters, p_w, p_i, β, and H.
H is the number of levels in the hierarchical system
p_w is the probability that the access is a write access.
p_i is the incremental probability. For an H levels hierarchical bus system, we assume that the reference by any processor is a hit with a probability $1-(H+1)p_i$.

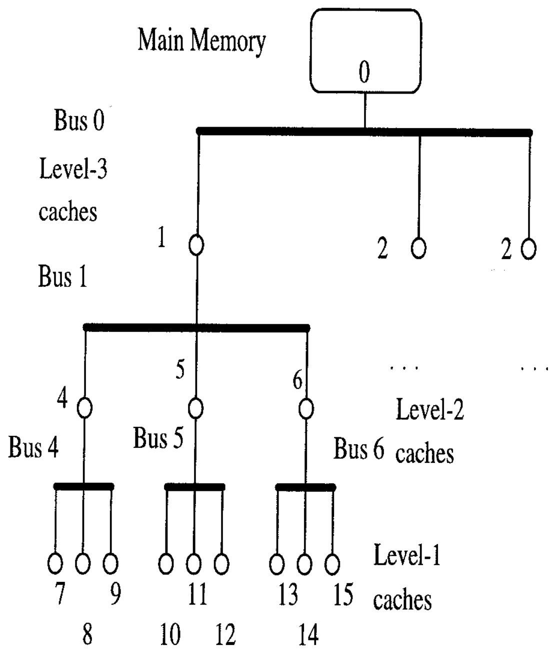

Figure 1: Hierarchical Shared Bus Multiprocessors

We assume that the reference is a miss, but the requested block resides in the same level-1 cluster as the requesting processor to be $1 - Hp_i$. Similarly, the probability that the requested block resides in the same level-2 cluster as the requesting processor is $1-(H-1)p_i$, ... and so on. Finally the probability that the referenced block reside in any cache (not the main memory) is $1-p_i$. Moreover, for every processor, we have a list of the Most Recently Used blocks (MRU list). If the reference is a hit, then there is a probability β that the reference is to one of the blocks in the MRU list, and with a probability $1-\beta$ to a block in the cache but not on the MRU list.

By using this model, we can change the degree of coupling between the processes as well as the write probability in order to test our proposed protocol under a wide range of workload.

4 Proposed Protocol

In this paper, we propose and analyze a single copy cache coherence protocol for hierarchical bus systems. We assume a tree hierarchy of single buses as shown in Figure 1. Hierarchical systems have proven to be a very effective, and scalable way of extending the number of processors in a multiprocessor system. In such a system, adding more processors means adding one or more buses which increase the total bandwidth of the system. Hierarchical systems are also suitable for the connectionist models of computations [FeB82], which require a very large number of processors and its applications are hierarchical in nature [GhH89]. One other advantage of hierarchical buses is it combines the simplicity of a single bus cluster with the scalability of multistage interconnection network; thus avoiding the bottleneck of connecting a large number of processors to a single bus.

As shown in Figure 1, this system consists of processors each with its own cache, we call these caches *processor's cache* or *level-1 cache*. At level-1, each k processors are attached to a single bus forming a first-level cluster. Every k first-level clusters are attached via k caches to a single bus (we call this *intermediate or shared cache*) forming a second-level cluster, and so on. The number of processors in the system is k^H where H is the number of levels in the system.

The first level caches behave just like ordinary cache. The idea is to maximize the hit ratio by predicting the next reference and trying to get the data to the cache before it is referenced. The function of the intermediate caches depends on the protocol used. In this paper we present three variations of a single copy cache coherence protocol. In BSCP (Basic Single Copy Protocol) data blocks are allowed to reside only in the processor's caches, the intermediate caches hold information about the blocks in the descendant caches in order to help locating any requested block quickly. In ESCP (Extended Single Copy Protocol) data blocks may reside anywhere in the system, thus the intermediate caches hold data blocks as well as information about the blocks in the descendant caches. In SMCP (Single Multiple Copies Protocol) we allow multiple copies of the same block within a single level-1 cluster only. In the same cluster, a snoopy cache coherence protocol is used to guarantee the consistency of the data.

Notice that because the protocol is a single copy protocol, the *inclusion property* has a different meaning in this context. Instead of saying that a cache at a certain level contains copies of all the blocks in the descendant caches, we modify this to state *an intermediate cache at a certain level contains information about the location of all the blocks in all the descendant caches.* This information could be in the form of a table with the block tag, and a number i $1 \leq i \leq k$, to indicate that the required block is in the i^{th} subtree rooted at this cache.

The Extended Single Copy Protocol (ESCP) seems suitable in cases where there is a large amount of sharing between processors, especially for some synchronization primitives that are frequently accessed by more than one processor. In order to avoid ping-pong effect, when more than one processor make more than one reference to the same data block, We allow the block to reside in an intermediate cache, the block could be placed in a cache that is as close as possible to the two requesting processors (root of the shortest subtree containing these two processors). In order to successfully manage such a scheme, we have to know in advance which block will be shared between more than one processor in the future. This is not always possible, and we have to depend on a simpler though less effective measures.

The Single Multiple Copy Protocol (SMCP), allows more than one copy of any block to reside simultaneously in the same level-1 cluster, but could not reside in more than one single level-1 cluster. A snoopy cache coherence protocol is used in each level-1 cluster to guarantee the consistence of the data.

4.1 BSCP Basic Single Copy Protocol

In the basic protocol, we assume a system like the one in Figure 1. Each processor fetches instruction and data from its own cache. In case of a cache miss, the processor sends a request to get the missing block. The request travels up the hierarchy. It checks every intermediate cache it encounters to see if the requested block is located in the subtree rooted at this particular intermediate cache. If the block does not exist, the request travels one level up, and continue searching for the block. If the block arrives at the main memory without locating the tree, the block is fetched from the main memory and returned back to the requesting processor. If the request located the block in a subtree rooted at a specific cache, it start descending down this subtree till the block is located and sent to the requesting processor.

One of the problems we encountered is as follows: A request is traveling up the tree looking for a specific block. After checking a specific cache, it indicates that the block does exist in the i^{th} subtree of this node. The request is enqueued waiting for this bus to access the lower level cache in the i^{th} subtree. While this request is waiting for the bus, the block is removed from its location by another request. After the original request capture the bus, and try to locate the block, it does not find it. In this case, our protocol sends a negative acknowledgment NAK to the requesting processor. After receiving the NAK, the requesting processor tries again without any delay (The delay the NAK encountered in its way to the processor is considered enough for the block to settle somewhere in the network). This is similar to the problem of transient blocks mentioned in [AnB93]. Our solution in this paper indicates an upper bound on the delay, i.e. any proposed solution to deal with this problem will result in more efficient protocol. we chose to deal with the worst case scenario because we are concentrating on the idea of a single copy protocols in hierarchical systems rather than the implementation details.

If an access results in a read or write miss, and the cache is full, then an LRU replacement policy is assumed and one block is chosen either to be written back to the memory if modified (dirty), or discarded if not modified (clean). The state of the requested bock will be the same as in the source cache.

Another point we considered, is the priorities in accessing the bus. In our simulation we assumed three different priority levels. The lowest priority is given to a request from a cache for a block that traveling up in the network. The second highest priority is given to a cache request traveling down the hierarchy. The highest priority is assumed for a moving data block, a write-back data block, a negative acknowledgment, or adding or removing a directory entry.

4.2 ESCP Extended Single Copy Protocol

In this protocol, we allow the data blocks to reside anywhere in the system, i.e. intermediate caches hold data blocks as well as information about blocks in the descending caches. The goal of this modification is to avoid ping-pong effect, where a block is referenced by 2 processors simultaneously. According to BSCP, this block will be bouncing back and forth between these two processor's caches.

In order to determine the blocks that are likely to be shared, (simultaneously accessed by more than one processor), We maintain for every cache a list with the α Most Recently Used MRU blocks as we explained before (in our simulation we set α to 8). If processor i issued a memory request and could not be found in its local cache, the request propagates up in the network till the block containing the requested data is located. The block could be in one of four locations, the action taken depends on the location of the requested block.

1 The block is found in the main memory: In this case, the block is transferred to processor i cache.

2 The block is found in processor j cache, but not in MRU_j: In this case, the block is transferred to processor i cache.

3 The block is found in processor j cache and is in MRU_j: then, according to the principal of locality, processor j will reference this block in the near future with a high probability. In this case, the block is transferred to the caching switch which is the root of the subtree that contain both processors i and j as leaves.

4 The block is found in an intermediate cache. In this case the block is transferred to processor i cache in the second consecutive reference by the same processor (i).

The actions taken if the block is in locations 1 or 2 are self explanatory, and simply state that the requested block is brought to the requesting processor cache.
If the block is in location 3 (*i.e.* in MRU_j), then processor j will probably request this block in the near future, moving it to cache i will result in ping-pong effect, we want this block to be on an equal distance from both processor i and processor j, that is why we move it to the root of the subtree containing both processor i and processor j.
If the block is in location 4, there are two possible reasons for any block to be in this location; either there are two processors actively referencing this block, or because it was brought to this location some time ago and no processor is actively referencing it. That is why we prefer to send the requested memory word and not the entire block on the first reference, but if the same processor (i) accesses the same block twice in a row, the block is moved to processor i's cache.

4.3 SMCP Single Multiple Copies Protocol

In this protocol we allow multiple copies of the same block to reside in multiple caches in the same level-1 cluster, thus decreasing the access time if a block is shared between two processors in the same cluster. In this protocol, data is allowed only in the processor's cache. The intermediate caches hold information about the location of each data block in the tree rooted at this particular intermediate cache except level-2 caches. Level-2 caches contain tables with two fields per entries. The first is the block number for all the block in the level-1 cluster. The second entry is the number of copies of this block in level-1 cluster. A snoopy controller monitors the bus and changes the number of copies for each block according to the transaction on the bus.

Each block in any processor cache could be in two states; either SHARED, or EXCLUSIVE. A shared block means that there are more copies of this block in the same cluster. Exclusive means it is the only copy in the cluster. Also each block could be clean (not modified), or dirty (modified). The protocol can be best described by following the actions of the snoopy controllers in 2 cases. Processors read or write, and requests from another processors. The action taken by the snoopy controller in case of read or write (hit or miss) can be described as

1 **Read hit** In case of a read hit, the processor goes ahead and read the requested word.

2 **Write hit** If the block is in Exclusive state, the processor goes ahead and writes. If the block is in a Shared state, the processor put an invalidate signal on the bus (as a result all other caches with a copy of the block set their copy to Invalid, and the level-2 cache sets the number of copies for this block to 1). Then, the block is modified.

3 **Read miss** The processor puts a request on the bus, if there are another copies in the cluster, one of the processor with a copy supply the block to the requesting processor, and the level-2 cache increment the number of copies for this block. If no other copies exists in the cluster, the request travels up the hierarchy to locate the block as in BSCP.

4 **Write miss** Similar to a read miss followed by a write hit.

The actions by the snoopy controllers in response to requests from other processors depends on the type of request, and can be described as follows.

1 **Invalidate** If the request is invalidate, the controller invalidates the copy of the block.

2 **Read** if the request is to read from a local (in the same cluster) processor, the controller supply a copy of the block on the bus, and sets the block state to shared. The level-2 cache increment the number of copies for this block. If the request is from a remote processor, the controller puts the block on the bus, and invalidate its own copy. The level-2 cache invalidate (purges) its entry for this block.

3 **Write** The controller puts the block on the bus, and invalidate its own copy. Level-2 cache sets the number of copies to 1 if the request is local. Otherwise, it invalidates its entry for this block and copies it to a higher level cache.

4 **Replace** If a block is to be replaced, the actions taken depends on the state of the block.

- Shared: The block is discarded and the higher level cache decrement the number of copies for this block.
- Exclusive: the block is written back to the main memory, and level-2 cache invalidates its own entry for this block

5 Simulation Results

We have simulated the system in Figure 1 using CSIM [Sch90], a process oriented discrete-event simulation package for use with C language. The number of processors is 27. We have assumed a total memory size of 1MBytes, a cache size of 128 blocks, a block size of 16 bytes, and a MRU list of size 8. Figure 2 shows the average access time for our BSCP and the protocol in [AnB93] for a probability of write 20% and $\beta = 0.8$. Figure 3 compares the performance of our three proposed protocols for a probability of write 20%, and $\beta = 0.8$. Figure 2 shows that our BSCP outperforms the directory based protocol proposed in [AnB92]. Figure 3 shows that the BSCP outperforms ESCP and SMCP. We explain this to the low probability of sharing between processors, thus making the overhead of moving the block to higher caches outweigh the benefits. It is remain to be seen if the same pattern will hold for real program traces instead of synthetic ones Figure 4 is the same as Figure 3 but with a probability of write 50%. In Figure 4 we notice that the performance of ESCP is approaching that of BSCP due to the increase of the write references. Figure 5 compares the three proposed protocols for $\beta = 0.2$, and $p_w = 0.2$. Figure 6 is the same as Figure 5 but with $p_w = 0.5$.

6 Conclusion

In this paper we proposed two single copy cache coherence protocols and a third protocol that allows a limited number of copies to exist provided that the copies are in the same level-1 cluster. We used a synthetic workload that can emulate a wide range of reference patterns. Our simulation shows that the performance of our three protocols exceeds the performance of the directory based protocol proposed in [AnB93] for a wide range of workload.

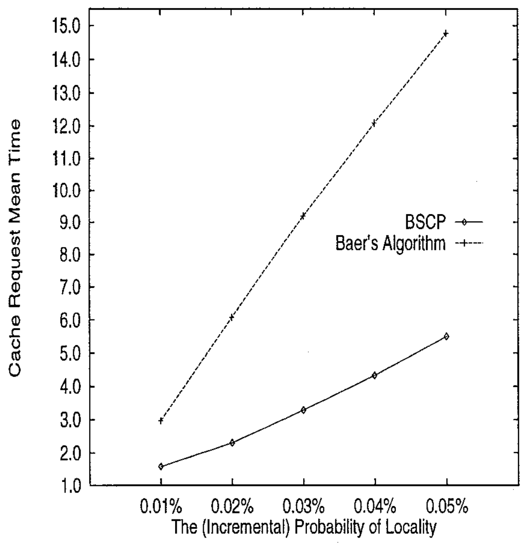

Figure 2: Average access time vs. the incremental probability for $\beta = 0.8$, $p_w = 0.2$

REFERENCES

[AgS88] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, *An Evaluation of Directory Schemes for Cache Coherence* "Proceedings of t he 15th Annual International Symposium on Computer Architecture," 1988, pp. 280-289.

[AnB92] C. Anderson and J.-L. Baer, *A Multi-Level Hierarchical Cache Coherence Protocol for Multiprocessors,* Technical Report number 92-10-04, Department of Computer Science & Engineering, University of Washington, 1992.

[AnB93] C. Anderson and J.-L. Baer, *A Multi-Level Hierarchical Cache Coherence Protocol for Multiprocessors* ' 'Proceedings of the 7th International Symposium on Parallel Processing," Newport Beach, CA May 1993, pp. 142-148.

[ChF90] D. Chaiken, C. Fields, K. Kurihara, and A. Ararwal, *Directory-Based Cache Coherence in Large Scale Multiprocessors* "IEEE Computer," Vol. 23, No. 6, June 1990, pp. 49-58.

[ChK91] D. Chaiken and K. Kubiatowicz, *LimitLESS Directories: A Scalable Cache Coherence Scheme* "Proceedings of the 4th International Conference on ASPLOS," 1991, pp. 224-234.

[EgK88] S. J. Eggers and R. H. Katz, *A Characterization of Sharing in Parallel Programs and Application to Coherency Protocol Evaluation* "15th Annual International Symposium on Computer Architecture," 1988, pp. 373-382.

[EgK89] S. Eggers and R. Katz, *Evaluating the Performance of Four Snooping Cache Coherency Protocols* "Proceedings of the 16th Annual International Symposium on Computer Architecture," 1989, pp. 2-15.

[FeB82] J. A. Feldman and D. H. Ballard, *Connectionist Models and their Properties* "Cognitive Science," Vol. 6 No. 3, 1982, pp. 205-254.

[GhH89] J. Ghosh and K. Hwang, *Mapping Neural Networks Onto Message Passing Multicomputers* "Journal of Parallel and Distributed Computing," Vol. 6, No. 2, April 1989, pp. 1-19.

[Goo83] J. R. Goodman, *Using Cache Memory to Reduce Processor-Memory Traffic* "International Symposium on Computer Architecture," 1983, pp. 124-131.

[MiB89] H. E. Mizrahi, J.-L. Baer, E. D. Lazowska, and J. Zahorjan, *Extending the Memory Hierarchy into Multiprocessor Interconnection Network: A Performance Analysis* "Proceedings of the International Conference on Parallel Processing," 1989, pp. I-41-I50.

[OmA94] R. Omran and M. Aboelaze, *An Efficient Single Copy Cache Coherence Protocol for Multiprocessors with Multistage Interconnection Networks* "Proceedings of Scalable High Performance Computer Conference," Knoxville, TN May 23-25, 1994, pp. 1-8.

[Sch90] H. Schwetman, *CSIM User's Guide,* MCC Technical Report Number:ACT-126-90, MCC Austin Texas, 1990.

[Ste89] P. Stenstrom, *A Cache Consistency Protocol for Multiprocessors with Multistage Networks* "Proc. of 16th Annual International Symposium on Computer Architecture ," 1989, pp. 407-405.

[ThD90] M. Thapar and B. Delagi, *New Directions in Scalable Shared Memory Multiprocessor Architecture: Stanford Distributed-Directory Protocol* "IEEE Computer," Vol. 32, No. 6, May 1990, pp. 78-80.

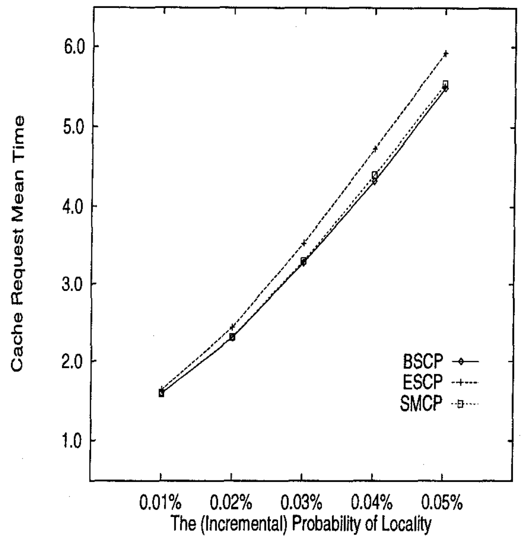

Figure 3: Average access time vs. incremental probability for $\beta = 0.8$, $p_w = 0.2$

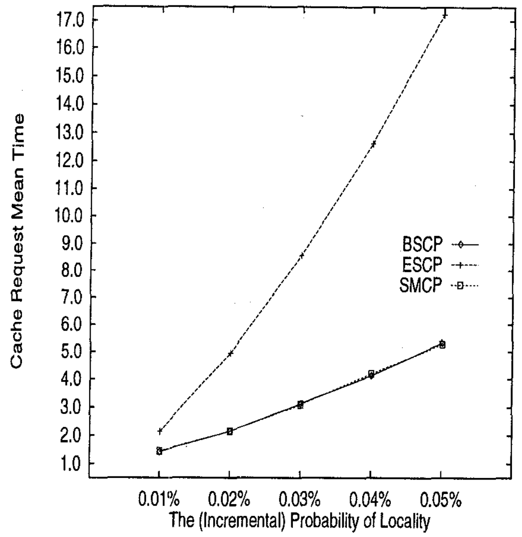

Figure 5: Average access time vs. incremental probability for $\beta = 0.2$, $p_w = 0.2$

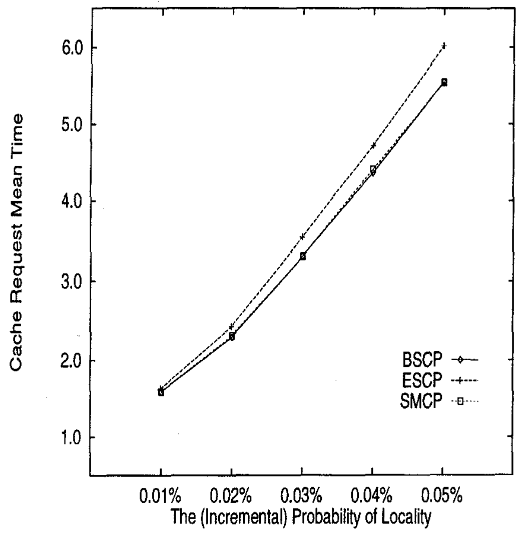

Figure 4: Average access time vs. incremental probability for $\beta = 0.8$, $p_w = 0.5$

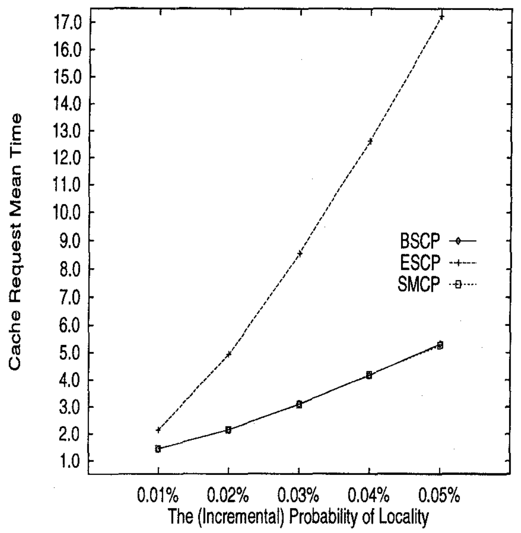

Figure 6: Average access time vs. incremental probability for $\beta = 0.2$, $p_w = 0.5$