An Innovative Implementation for Directory-based Cache Coherence in Shared Memory Multiprocessors*

Weisong Shi Weiwu Hu and Ming Zhu
Center of High Performance Computing, Institute of Computing Technology
Chinese Academy of Sciences, P.O.Box 2704-35, Beijing 100080, P.R.China
e-mail: {wsshi,hww,zm}@water.chpc.ict.ac.cn

Abstract

Directory-based cache coherence protocol is accepted as the common technique in large scale shared memory multiprocessors because of its scalability. Although it was extensively studied in the past, however, the memory overhead and long miss penalty entailed by this protocol are the major obstacles to scale for large scale multiprocessors. On the other hand, the ever-increasing cache line size makes the false sharing problem more serious than before, which will lead to high miss rate. Based on the scope consistency, we propose a lock-specific home-based cache coherence protocol. In this new cache coherence protocol, all directory memory overhead are eleminated completely and false sharing problem and high miss rate will be sloved greatly at the cost of small write notice buffer in each processor.

Keywords: Directory-based Cache Coherence Protocol, False Sharing, Scope Consistency, Memory Overhead.

1 Introduction

Distributed Shared Memory(DSM) systems have gained popular acceptance by combing the scalability and low cost of distributed system with the ease of use of single address space. Many research and commercial parallel systems are built out of this architecture. In order to tolerate the long remote access latency and exploit the advantage of prefetching, cache coherence is an accepted requirement in large scale multiprocessors[14]. Bus-based snoopy coherence protocol and directory-based coherence protocol are widely used in shared memory multiprocessors. Unfortunately, the bus can only accommodate a small number of processors and such machines are not scalable. For large-scale multiprocessors, we need a scalable interconnection network such as mesh, torus, which makes snoopying impossible.

Directory-based cache coherence was first proposed by Tang[17] and Censier and Feautrier[4] . The basic idea is keeping track of the information of each memory block in a directory, and all requests to the cache block will be sent to it's corresponding directory controller first, which will determine the related actions. The general architecture of distributed shared memory multiprocessors is shown in Fig 1.

For directory schemes to be successful for scalable multiprocessors, they must satisfy two requirements [7]. The first is that the bandwidth to access directory information must scale well with the number of processors. This requirement can be achieved by distributing the physical memory and the corresponding nodes, and by using a scalable interconnection network. The second requirement is that the hardware overhead of using a directory scheme must scale linearly with the number of processors. The most important component of the hardware overhead is the amount of memory storing the directory informantion. Up to date, this requirement are gained extensively study and many innovative ideas are proposed. The details are listed in the following section. However, the memory overhead remains a major obstacle for the scalability of the directory-based coherence protocol.

*The work of this paper is supported by the CLIMBING Program and the President Young Investigator Foundation of the Chinese Academy of Sciences.

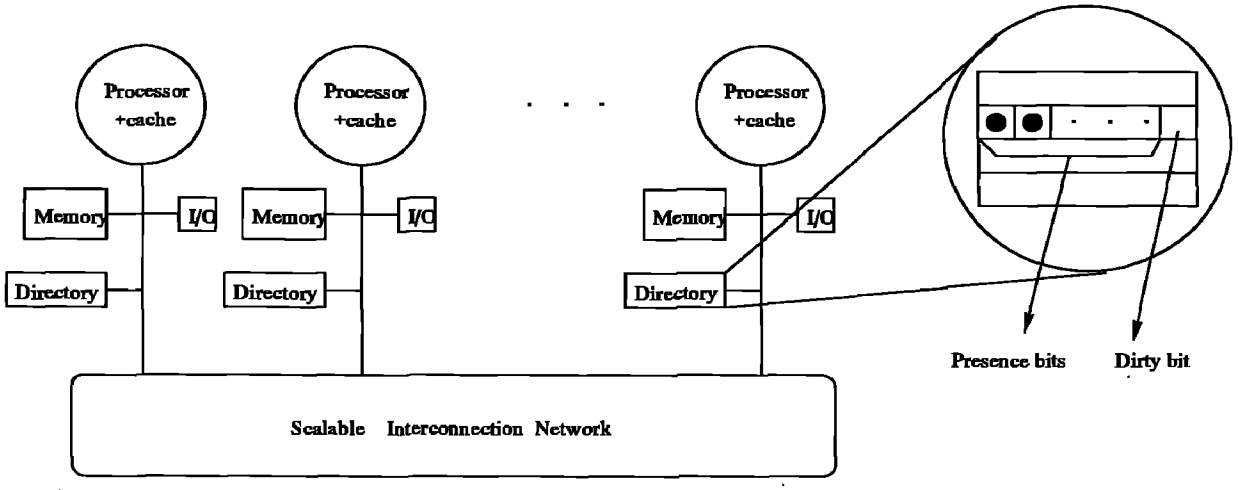

Figure 1: Basic multiprocessor organization and directory scheme.

Furthermore, from the state transition diagram of directory-based coherence protocol as shown in [14], we find that 3-hop communication is required frequently when write miss occurs, even when a write operation hits on read-only cache blocks. Thus, it will result in long miss penalty. Under release consistency, this miss penalty can be tolerated by overlaping the invalidation procedure with computation procedure, however, the processor on which write miss occurs must wait for the acknowledgement from the directory controller[15], thus the long miss penalty remains a problem to be solved.

On the other hand, the ever-increasing size of cache line makes false sharing problem become more serious than before. False sharing is the sharing of cache blocks without actual sharing of data. It occurs because cache blocks contain more than one data item. Whenever in write-invalidate or write-update protocols, more traffic and miss rates are caused by false sharing so that the whole system performance degrade significantly[18].

However, in software DSM systems, such as TreadMarks[1], Munin[3] etc, the false sharing problem are sloved greatly because of using lazy release consistency and multiple writer protocol[1][3]. Therefore, in this paper, we propose a new cache coherence protocol which is based on the scope consistencyand is easy to implement in hardware.

The rest of this paper is orgnized as follows. The background and prior related work are listed in section 2. Our new innovative cache coherence protocol is presented in section 3. Some important issues of implementation are described in seciton 4. In the last section, concluding remarks are provided.

2 Background and Related Work

2.1 Directory-based Cache Coherence Protocol

In directory-based cache coherence protocol, the most straightforward way to store directory information is as a bit-vector associated with each memory block. Each directory entry has one presence bit per processing node(thus named *full bit vector* scheme), and one dirty bit indicating ownership by a given processor(the one whose presence bit is also on), as shown in Fig 1. A good measure of the memory overhead of a directory scheme is the amount of storage used for directory information divided by the storage used for the memory blocks themselves. Assuming a 64-byte line, the directory overhead for 64-node is approximately 12.7%, for a 1024-node multiprocessors, it is approximately 200%. Therefore, the directory storage overhead clearly does not scale well to a large number of processing nodes in this scheme.

There are two ways in which the overhead of *full bit vector* scheme can be reduced. The first is to increase the cache line (block)size. However, as discussed in section 1, this will lead to false sharing problem. The second technique to reduce the directory overhead is to use clustering, such as DASH[13] and SGI Origin 2000 [12] . Nevertheless, both the optimizations reduce the overhead by a little factor. The total amout of

directory memory overhead is still proportional to $P \times M$, where P is the number of processoring nodes and M is the number of total memory blocks in the machine.

For larger-scale multiprocessors, there are two orthogonal ways to reduce the overhead of directory schemes. The first is to reduce the width of the each directory entry by not letting it grow proportionally to P. The representative schemes include: $Dir_iB, Dir_iNB, Dir_iCV_r, Dir_iSW$, Dir_iDP [5][16] and cache-based directory coherence scheme(such as SCI[8]). The second is to reduce the total number of directory entries by not having an entry per memory block, such as *sparse directories*.

Although many new directory orgnization schemes are proposed, however, the memory overhead remains a major obstacle for its scalability. On the other hand, the state transition remains complicated and false sharing problem are of great necessary to be solved. Therefore, some researchers proposed new consistency models for shared memory multiprocessors.

2.2 Delayed Consistency

Delayed Consistency was proposed by Dubois et.al in 1991[6], which means the effect of out-going and incoming invalidations or updates are delayed. They introduced two delayed protocols derived from Censier and Feautrier's directory scheme. One is *receive delayed protocol*, the other is *send and receive delayed protocol*. Both these two protocols are delayed implementation under a weakly ordered memeory istency model, called release consistency. The basic idea of receive delayed protocol is that when an *invalidation* signal is received by a cache, the invalidation does not need to reach the cache until the next *acquire* operation is executed by the local processor. The behaviour is still correct because the programming model in weakly-ordered systems forbids accesses to a shared writable data outside a critical section. In send and receive delay protocol, the processor can delay sending the invalidation to other sharers until next release synchronization operation.

Delayed consistency solves the false sharing problem, and significant reductions in the miss rate can be obtained with the addition of a stale bit, and further reduction was observed by adding a small ISB(invalidation send buffer).

In received delayed protocol, each processor executing write operation send the invalidation immediately, and waits for the acknowledgement from the directory controller. The addtional hardware is a stale bit with each cache line. Although false sharing problem is reduced considerably, the direcotory overhead remains a bottleneck. Furhtermore, the complicated state transtion makes this protocol difficult to implement.

In send and receive delayed protocol, more than two processors can write the same cache block simultaneously (i.e., multiple writers protocol). There are only one owner allowed to exist in the system at any time. So when other *keeper* want to remove its ISBs, it must obtain the ownership first, which will result in communicating with others. Therefore, the frequency of comunnication is similar to that of traditional protocol. At the same time, the directory overhead and complicated state transtion remain serious problems.

2.3 Lazy Release Consistency

In [11], Kontothanassis et.al proposed to implement lazy release consistency[1] in haredware multiprocessors. In fact, the basic idea of their scheme is similar to delayed consistency discussed above. The difference between these two protocols exists in which level to implement. Lazy release consistency is implemented in directory level(i.e., each directory has 4 states, as shown in Fig 2(b).), while the Dubois' implemented it in cache line level(i.e., each cache line has 4 states, as shown in Fig 2(c)). The lazy release consistency adopted in [11] combines the multiple writer protocol and receive delayed protocol. The home is keep up to date, so no ISB is needed. Although this hardware implementation scheme reduces the miss rate and false sharing transitions greatly, it need more directory overhead than traditional protocol.

3 New Cache Coherence Protocol

From above discussion, we find that some improvements in reducing the directory overhead do not help slove the false sharing problem and reduce the high miss rate, and some improvments in reducing miss rate

[1]Here, lazy release consistency is defined by authors in [11], which is a little bit different from the definitin of Keleher et.al in [10]. Therefore, we named the definition of Keleher etc standard lazy release consistency.

do not help reduce the directory overhead, even to the contrary. In this section, we will introduce a new consistency model, then propose an innovative coherence implemented scheme which will solve above two problems efficiently.

In *standard lazy release consistency*, the requesting processor will invalidate all the pages which were written by another processors before this acquire operation according to happen-before-1 order. However, different locks are not discriminated to treat with, which results in many useless invalidations and high miss rate on shared data. This disadvantage of lazy release consistency is overcomed by discriminating the different locks in *scope consistency*[9], which means the acquiring processor invalidates the pages associated with this lock only, other pages which is protected by another synchronization objects are kept valid. With these two relaxed consistency models, the false sharing problem will be avoided greatly.

Furthermore, multiple writer protocol is very helpful to reduce the miss rate, especially when write hit on read-only occurs. Therefore, our new cache coherence protocol combines the advantages of these two techniques, adopts write-invalidate based multiple writer protocol under the scope consistency.

In our new cache coherence protocol, we assume that synchronization variables must be stored in specialized regions of shared memory since it has been implemented in many new generation machines, such as SGI Origin2000[12], each lock has a manager. This manager has the responsbility to keep the write notices modifed in the critical section protected by this lock, which is similar to the standard lazy release consistency. The home node does not need to maintain a directory for each block. The home is usually kept up to date. The cache line state transition graph is shown in Fig 2(a). Only three states are required, which is simpler than that of prior works.

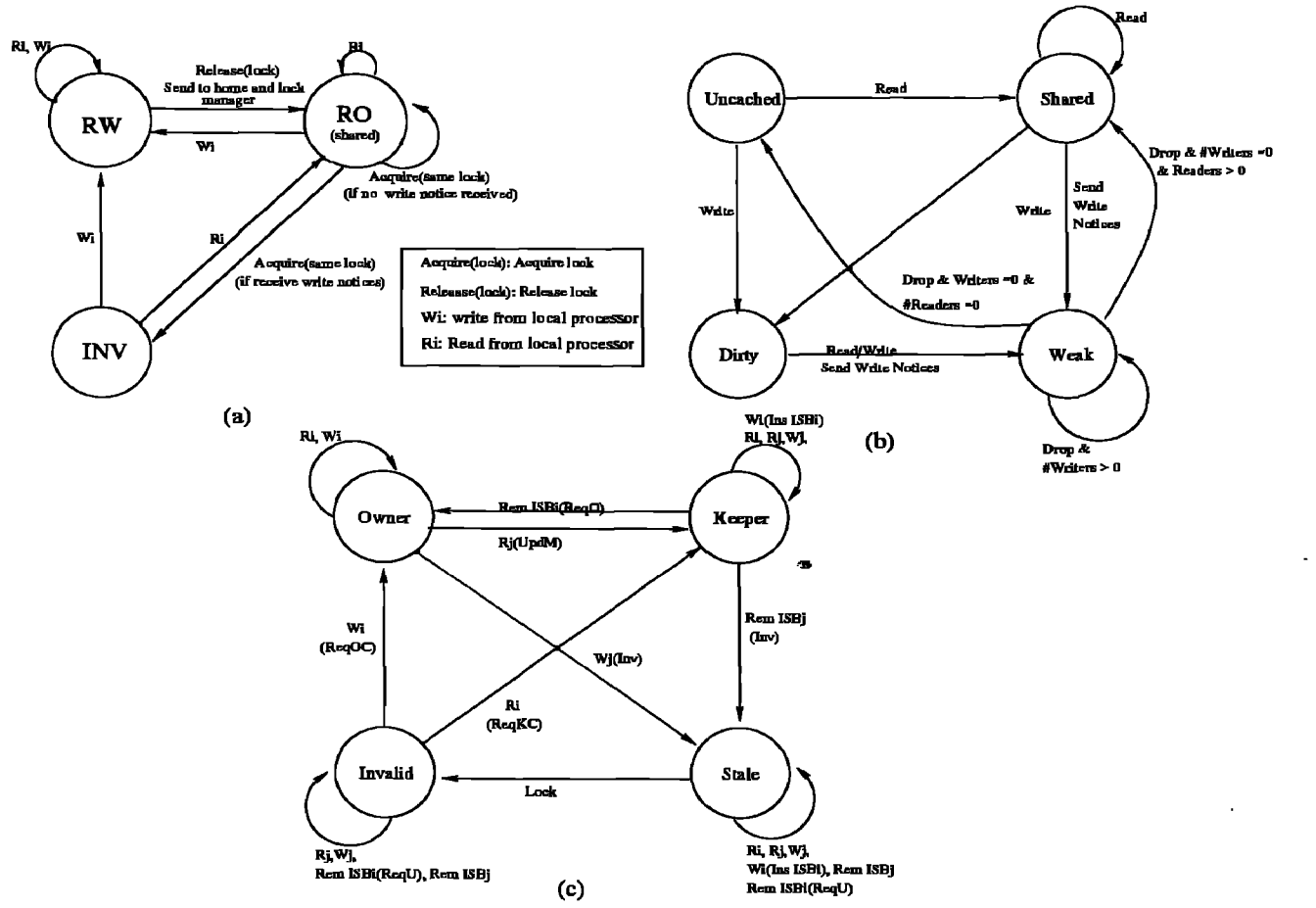

Figure 2: The state transition graph of cache coherence protocol(a) our new scheme, (b) Kontothanassis's lazy release consistency scheme, (c) Dubois's send and receive delayed consistency scheme.

Cache Block States

RW —- the cache block is readable and writable;

RO —- the cache block is readable;

INV —- the cache block is invalid.

No directory is needed, so no system directory states are required.

Memory Commands

(a) *Issued by a cache line to the home memeory controller.*

1. RequestData: send the request about a cache line to corresponding home node.

(b) *Issued by a home memory controller to the cache line.*

1. ReturnData: send the corresponding memory block to requesting processor.

(c) *Issued by a cache line to the lock memory controller(lock manager).*

1. RequestLock: ask for the ownership of the lock.

(d) *Issued by a lock memory controller(lock manager) to the cache line.*

1. Wait: ask the requesting processor to wait for another processor's releasing this lock.
2. ReturnLock: send the ownership of the lock to the requesting processor and associated write notices.

Cache algorithm

For the various types of cache accesses, the cache controller takes the following actions.

1. Read hit: no action is taken.
2. Write hit: no action is taken(including write hit on read only cache line.).
3. Read miss: send a request to home node to obtain a copy.
4. Write miss: send a request to home node to obtaib a copy.
5. Replacment: write the cache block back to its home node, and inform the lock manager if the state of this cache line is RW.
6. Acquire(lock1) : send the request to lock1's manager.
7. Release(lock1): send the addresses of those cache blocks that modified within above critical section to lock1's manager and all the modified values to their corresponding home nodes.

Compare our new cache coherence protocol with prior related works, our scheme will has following advantages:

1. Solve the directory overhead substantially, which makes this scheme overcome the scalability limit.
2. The different synchronization objects are discriminated to treat with, which make the number of invalidation reduce greatly, so the miss rate avoided greatly.
3. The cache state transition diagram becomes simpler than prior works.
4. The latency of each miss is 2-hop communication substantially, no 3-hop communications is required.
5. False sharing problem is avoided by using multiple writer protocol.

However, our new will incur following disadvantages:

1. At release operation, the processor will wait longer than before.

2. Each lock manager require some memory overhead to hold the write notices for that lock.

Certainly, in order to reduce the memory overhead asscociated with each lock manager, we will fix the size of write notice buffer in each lock manager. If this write notice buffer is overflowed, the lock manager will start a global invalidation operation. The organization of those write notice buffers is shown in next section. Whether the advantages of our new scheme can tradeoff the disadvantages entailed from it, we will demonstrate it by simulation in the near future.

4 Important Issues of Implementation

The most important implementation issues in our new cache coherence is how to memorize the write notices. In fact, our write notice is simpler than that of TreadMarks. When one processor write to a cache block, it does not need to create a twin and diff. An alternative implement scheme is as follows.

1. Acquire a lock.
2. Write to a cache block, allocate a item in write notice buffer(addition hardware) and fill the address and content of this cache block into this item.
3. If possible, send the address and data of above cache block to lock manager and home respectively, which overlap the communication and compution[a].
4. Release the lock, and write all the data in write notice buffers to their corresponding home nodes, and sending addresses of these cache blocks to their lock's managers.(This processor must wait until the acknowledgements from the corresponding counterparts).
5. Invalidate all the items in write notice buffer, which make them reusable in the future.

[a]This operation is optional if there is protocol processor in the system.

In each lock manager, it will have a specialized write notice buffer which memorize the address of cache block only. This write notice buffer should contain no more than a few entries in order to reduce its memory overhead. Fig 3 shows the memory organization of two kinds of write notice buffers. Fig 3(a) shows the write notice buffer used by general processor during it's execution in critical section. Fig 3(b) shows the write notice buffer used by lock manager. In fact, the write notice buffer in general processor can be implemented by write buffer in modern microprocessors. For example, before the write operation retired from *active list* or *instruction window*, it must be sent to its home node.

Address 1	data
Address 2	data
. . .	. . .
. . .	. . .
Address n	data

(a)

Address 1
Address 2
. . .
. . .
Address n

(b)

Figure 3: The organization of write notice buffer(a) in general memory controller;(b) in lock manager.

When a processor acquires a lock, it send a message to lock's manager. The lock manager returns all those addresses associated with this lock to this acquiring processor. After receiving these addresses, the

acquiring processor invalidates it's corresponding cache blocks. When there is no free space left in write notice buffer, the lock manager must execute a global operation to invalidate all the addresses memorized in it's write notice buffer. This operation will take a long time so that it can not occur frequently in real life. Forturnately, some statistics show that 80% applications adopt read-write sharing pattern. Therefore, this operation will not occur so frequently. Certainly, whether this new scheme better than traditional coherence protocol or not is a tradeoff between time and space. If the time is acceptable, we belive that our new scheme is better than others. That is our future work.

5 Summary

In this paper, we analyze the disadvantages of direcory-based cache coherence protocol first. We also point that the high miss rate and false sharing problem are of great importance in future generation computers. Based on the analysis of prior works, we propose a new cache coherence implementation scheme which is based on scope consistency and multiple writer protocol. Our new scheme requires only three states to implement multiple writer protocol, which is a great improvement to prior works. Our new scheme solves the directory memory overhead substantially. The second effect is reducing the miss penalty to 2-hop communication. We will compare our new scheme with prior works by simulation in the future.

References

[1] Cristiana Amza,S. Dwarkadas,Pete Keleher,Alan L.Cox and Willy Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations.*IEEE Computer*,29(2):18-28,February 1996.

[2] B.N.Bershad,M.J.Zekauskas,and W.A.Sawdon. The Midway Distributed Shared Memory System, In *Proc.of the 38th IEEE Int'l Computer Cong. (COMPCON Spring'93)*, pp.528-537,February 1993.

[3] J.B.Carter,J.K.Bennet,and W.Zwaenepoel.Implenentation and Performance of Munin. In *Proc.of the 13th ACM Symp.on Operating Systems Principles (SOSP'91)*, pp.152-164,October 1991.

[4] M.Censier and P.Feautier. A New Solution to Coherence Problems in Multicache Systems. *IEEE Transactions on Computers*, C-27(12):1112-1118, December 1978.

[5] David E.Culler, J.P.Singh with Anoop Gupta. *Parallel Computer Architecture (alpha version). available in http://www.cs.berkeley.edu/~culler/book-alpha.html.*

[6] Michel Dubios. Jin Chin Wang, Luiz A.Barroso, Kangwoo Lee and Yung-Syau Chen. Delayed Consistency and Its Effects on the Miss Rate of Parallel Programs. In *Supercomputing91*. pp.197-206.

[7] Anoop Gupta, Wolf-Dietrich Weber, and Todd Mowry. Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In *ICPP'90,* pp.I-312-I321.

[8] David Gustavson. The Scalable Coherence Interface and Related Standards Projects. *IEEE micro.* 12(1), pp:10-22, February 1992.

[9] L.Iftode,J.P.Singh,and K.Li. Scope Consistency:a Bridge between Release Consistency and Entry Consistency. Technique Repopt TR-509-96, Princeton, NJ, February 1996.

[10] Pete Keleher, Alan L.Cox, and Willy Zwaenepoel. Lazy Release Consistency for software Distributed Shared Memory. In *the 19th ISCA'92*, pp.13-21.

[11] Leonidas I. Kontothanassis, Michael L.Scott and Ricardo Bianchini, Lazy Release Consistency for Hardware-Coherent Multiprocessor. In *Supercomputing'95.*

[12] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In Proceedings of the 24th Annual International Symposium on Computer Architerture(ISCA'97). Denver, CO, USA. pp.241-251.

[13] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta and John L. Hennessy. The Directoty-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th International Symposium on Computer Architecture*, 1990, pp.148-159.

[14] D.Patterson and John Hennessy. *Computer Architecture: A Quantitative Approach.* Second Edition. 1996.

[15] Weisong Shi, Weiwu Hu and Zhimin Tang. An Interaction of Coherence Protocols and Memory Consistency Models in DSM Systems. *To appear in ACM Operating Systems Review*, pp.48-61, October 1997.

[16] Richard Thomas Simoni, Jr. *Cache Coherence Directories for Scalable Multiprocessor.* Ph.D thesis. Stanford University, March 1995.

[17] C.K.Tang. Cache Design in the Tightly Coupled Multiprocessor System. In *AFIPS Conference Proceedings, National Computer Conference, NY*, pages 749-753, June 1976.

[18] J.Torrellas, Monica S.Lam and John Hennessy. Shared Data Placement Optimization to Reduce Multiprocessor Cache Miss Rates. In *1990 International Conference on Parallel Processing.* pp.II-266-II-270.