Lecture Notes in Computer Science 591

H. P. Zima (Ed.)

Parallel Computation

First International ACPC Conference
Salzburg, Austria, September/October 1991
Proceedings

Springer-Verlag

BIBLIOTHEQUE DU CERIST

Lecture Notes in Computer Science 591
Edited by G. Goos and J. Hartmanis

Advisory Board: W. Brauer D. Gries J. Stoer

BIBLIOTHEQUE DU CERIST

H. P. Zima (Ed.)

Parallel Computation

First International ACPC Conference
Salzburg, Austria, September 30 - October 2, 1991
Proceedings

BIBLIOTHEQUE DU CERIST

Cc 01-591

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Digital Equipment Corporation GmbH Campusnahes Forschungszentrum (Vienna)
IBM Austria (Vienna)
Sony Austria GmbH (Anif, Salzburg)
MASPAR Distributor AG (Zürich-Oberengstringen, Switzerland)
Intel Corporation Ltd. (Swindon, UK)
nCUBE Deutschland GmbH (Munich)
Bacher Electronics GmbH (Vienna)
SiliconGraphics Computer Systems (Grasbrunn-Neukäferloh)
Cray Research GmbH (Munich)
Cray Research Inc.(USA)
Meiko Limited (Bristol, UK)
Parsytec (Aachen)
Floating Point System GmbH (Riemerling)
Convex GmbH (Frankfurt)
Control Data GmbH (Vienna)
Emco Maier and Co. (Hallein)

Vienna, March 1992 Hans P. Zima

BIBLIOTHEQUE DU CERIST

Contents

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

Scalable Cache Coherence for Shared Memory Multiprocessors

Manu Thapar
Digital Equipment Corporation
and
Stanford University

Bruce A. Delagi
Sun Microsystems
and
Stanford University

Michael J. Flynn
Stanford University

701 Welch Road, Palo Alto, CA 94304, USA

BIBLIOTHEQUE DU CERIST

Abstract

This paper presents a performance analysis of a new directory based cache coherence protocol. We compare the fully mapped centralized directory protocol with a distributed directory protocol developed by us. The distributed directory protocol is based on a linked list of caches and is more scalable in terms of cost and performance. It does not require the network to preserve the order of messages and allows adaptive routing so that network performance may be more robust. Simulation results show that the distributed directory protocol has better performance than the centralized directory protocol for the benchmarks we have analyzed.

1 Introduction

In a shared memory multiprocessor system, each processor usually has an associated cache. If these multiple caches are allowed to simultaneously have copies of a given memory location, a mechanism must exist to ensure that all copies remain consistent when the contents of that memory location are modified. This is known as the cache coherence problem, which is an important and well known problem in shared memory multiprocessors. "Snoopy" cache coherence protocols are well understood for bus-based shared memory architectures [2]. These protocols require that each cache watch all traffic on the bus and take appropriate action for addresses that are present in that cache. Addresses are, in effect, transmitted to each cache by global broadcast. The shared bus limits the number of processors to the number that can be connected to the bus without saturating it. To support *scalable* shared memory architectures, the cache coherence protocol must work in the absence of a global broadcast mechanism. Centralized directory based schemes [1, 4] are a possible solution in this environment. More recently, protocols based on a linked list of caches have been proposed [11, 8]. In this paper we compare the

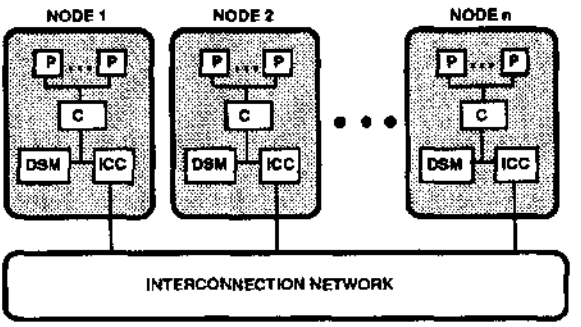

Figure 1: The basic architecture

BIBLIOTHEQUE DU CERIST

fully mapped centralized directory protocol [4] with a distributed directory protocol [11] that we have developed.

In the distributed directory protocol, the information about which caches have copies of the data is decentralized and distributed among the cache lines. Our implementation, like the fully mapped centralized directory scheme, tracks any number of cache copies and never requires invalidates to be sent to all caches in the system. It is scalable to larger systems and has better performance than the fully mapped directory based coherence scheme. In the fully mapped scheme, the size of the memory required to hold the state information is $O(MN)$, where M is the size of main memory and N is the number of caches. In our scheme, on the other hand, the size of the memory required to hold the state information is only $O(M \log N)$. We do not assume that the interconnection network preserves the order of messages and thus allow adaptive routing. The protocol also allows an efficient implementation of locks [11].

2 Centralized Directory Protocols

We assume a very general computing system structure in our description of the protocols. Figure 1 describes this basic architecture. Each node consists of one or more processing elements (P), a cache (C), an interconnect controller (ICC) and part of the distributed shared memory (DSM). The DSM includes the directory.

In the directory based protocols there is a directory "tag" associated with each line in main memory. This directory is used to hold information about which caches have copies of the line. In the fully mapped centralized[1] directory scheme, the directory has N valid (or "present") bits per line, where N is the number of caches. The amount of storage needed for the directory in the fully mapped scheme is thus $O(MN)$, where M is the size

[1]We use the term *centralized* since the information about caches that have copies of a memory line is located at one place. The directory tags are an extension of the lines in the DSM and are located on the same node as the corresponding lines in main memory.

of main memory. If a cache has a copy of the line, the present bit corresponding to that cache is set. The directory also has a dirty bit. If the dirty bit is set, only one of the caches can have a copy of the line.

On a read miss, the directory is checked to see if the block is dirty in another cache. If so, consistency is maintained by copying the dirty block back to the memory before supplying the data. The reply is thus serialized through the directory. To ensure correct operation, the memory line has to be "locked" by the directory controller until the write-back signal is received from the cache with the dirty block. No other coherency related operations on this line may be undertaken while a line is locked. If the line is not dirty in another cache, then data is supplied from the main memory and the corresponding present bit is set in the directory.

On a write miss, the central directory is checked to determine the state of the line. If the line is dirty in another cache, then the line is first flushed from that cache before supplying the data. Again, the reply is serialized through the directory. The memory line is locked while this is being done. If the line is clean in other caches, invalidate signals are sent to the caches. The memory line is locked until acknowledgements are received from the caches. The data can then be supplied to the requesting cache. Thus, if the line is present in one other cache on a write miss, four network operations are required before the write can be considered to be complete. These include:

1. The miss signal that is sent to the main memory.
2. The invalidate or write-back signal that is sent to the cache that has the data in clean or dirty state respectively.
3. The invalidate-acknowledge or write-back-data signal that is sent from the cache that has the data in clean or dirty state respectively.
4. The write-miss-reply is sent from the main memory to the requesting cache.

The serialization of responses through the directory and the locking of lines by the directory controller impacts the performance of the cache coherence scheme. Requests that arrive while a line is locked have to be either buffered at the directory, or else bounced back to the source to be reissued at a later time. If the requests are buffered at the directory, the network traffic is lower. However, if the buffer overflows, the requests still have to be bounced back. Requiring transactions to be serialized through the centralized directory (and the locking of lines while servicing a request that requires a coherency-related transaction) could make the directory a bottleneck.

To reduce the amount of storage required, a number of modifications to the above scheme may be made [1]. However, these modifications either require the implementation of an efficient broadcast mechanism contradicting our assumption about scalable systems, or may generate excess network traffic along with performance penalties. For example, one simple modification is to have i pointers per line in the directory. Each pointer may point to a cache that has a copy of the line. If more than i caches have copies of the line, a broadcast has to be done to *all* caches to service a write miss. The memory line has to be locked until all caches acknowledge the invalidation. Another alternative is to allow at most i caches to have copies of a line at the same time. In the case where a read

BIBLIOTHEQUE DU CERIST

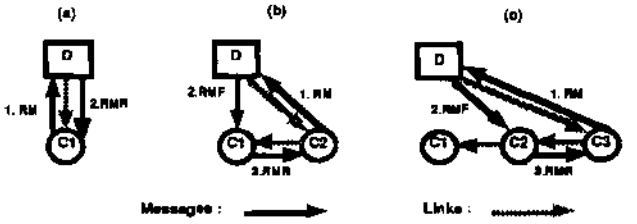

Figure 2: Linking of caches due to read misses

miss occurs when i caches have copies of the line, the directory has to invalidate one of the copies before the data can be supplied to the requesting cache. This might result in "thrashing" the line between caches.

The amount of memory required for the directory may also be reduced by caching the directory [7]. This technique may be used to further reduce the amount of memory required for the distributed directory protocol as well. In this paper we compare the distributed directory protocol with the fully mapped centralized directory protocol which has better performance than any of the centralized directory protocols that try to minimize the amount of memory required for the directory.

3 The Distributed Directory Protocol

In our distributed directory protocol, caches that share data are linked together in a list. Each line in the main memory and the cache has a cache-pointer field associated with it. This pointer can specify any cache in the system. The directory services a read or write miss request by changing the cache-pointer in the directory entry associated with the line to point to the requesting cache. A line in main memory is originally in state "absent" from all caches. Each request causes the value of the cache-pointer to be updated to point to the requesting cache. If the line is absent from all the caches, the main memory sends a reply. Otherwise the request is forwarded to the last cache to make a request for the same line.

In case of read misses, that cache replies to the requesting cache. The reply consists of the data and the address of the replying cache. The requesting cache sets its cache-pointer to point to the replying cache. A singly-linked list of caches that contain shared copies of the data is thus formed. Read misses require a maximum of three network operations regardless of the length of the linked list.

A line in cache memory is originally in state "invalid". A read or a write request from the processor causes the state to change to "writing-or-reading" and a read-miss or write-miss signal to be sent to the appropriate main memory module. On a read-miss-reply, the value of the cache-pointer is set to be the address of the object sending the reply. This causes a linked list of caches that contain the data in shared state to be formed. Figure 2 illustrates the process followed to set up the linked list. Consider the case where cache C1 has a read miss for a line followed by caches C2 and C3. As show in fig. 2(a), cache C1 sends a read-miss signal to the directory. The cache-pointer of the line in the directory

BIBLIOTHEQUE DU CERIST

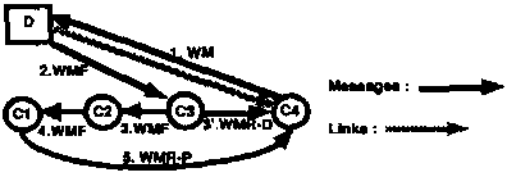

Figure 3: Invalidations due to write misses

is made to point to C1. Since no other cache has a copy of the line, the main memory sends a read-miss-reply to C1. When C1 receives the reply, the line is loaded into the cache in state "exclusive". Now, when cache C2 sends a read-miss to the directory, a read-miss-forward signal is sent to C1 as shown in fig. 2(b). The directory does not send a reply directly to C2 since C1 may have written to the line locally. The cache-pointer in the directory now points to C2. When C1 receives the forwarded signal, it changes its state to "shared" and sends a read-miss-reply to C2. The reply includes the data and the address of C1. When C2 receives the reply, it sets its cache-pointer to point to C1. Thus a linked list is formed. Fig. 2(c) shows how C3 gets linked into the list.

Write misses cause a write-miss signal to be sent to the directory. A line is allocated in the cache before the miss signal is sent. This line is used to buffer the write. Write buffering along with weak ordering [6] allows the processor to proceed immediately without stalling. A write is considered to be *issued* when a write-miss is sent by the cache. A write is considered to be *performed* when a write-miss-reply is received by the cache. A write-miss-reply may consist of two signals as in the example below. A *fence* [3] operation may be used to ensure that all writes that have been issued by a processor are performed before that processor is allowed to proceed. If a copy of the line is not present in any other cache, the main memory directly sends a reply. Otherwise, the copies of the line have to be invalidated before a reply can be sent.

Figure 3 shows the sequence of events that result when multiple caches have a copy of the line and C4 has a cache miss. The directory forwards the write miss signal to the old head (C3) pointed to by the cache-pointer and the cache-pointer is updated to point to C4. When C3 receives the write-miss-forward signal, it invalidates its copy and forwards the signal to C2. C3 also sends a write-miss-reply-data signal along with the requested data to the requesting cache C4. When C2 receives the write-miss-forward signal, it invalidates its copy and forwards the signal to C1. Since the cache-pointer of C1 points to the directory, it can be determined locally that C1 is the tail of the list and a write-miss-reply-performed signal is sent to C4 after the data in C1 is invalidated. C4 needs to receive both the write-miss-reply-data and the write-miss-reply-performed signals before the write can be considered to be performed.

In the distributed directory protocol, the information about which caches have copies of the data is distributed among the cache lines. The servicing of requests does not require any locking of lines as in the case of the centralized directory protocol. Direct cache-to-cache operations are used to send the replies and none of the replies have to be serialized through the main memory. The centralized bottleneck which is present in the centralized directory protocols is thus eliminated.

BIBLIOTHEQUE DU CERIST

A cache line would be in state "writing-or-reading" after a read-miss or a write-miss has been generated and before a read-miss-reply or a write-miss-reply has been received. If the line in the cache is in state "writing-or-reading" and a read-miss-forward or a write-miss-forward signal is received, the forwarded signal is stored in the cache-pointer field of the cache line. The state is changed to note that a forwarded signal has been stored. Such signals that are stored are called *pending signals* and are serviced when the reply to the local read or write miss is received. If multiple transactions for the same line are pending, the caches form a *distributed queue* of pending signals. The requests are thus serviced in a pipelined manner rather than causing any bouncing of signals or contention at the directory as in the case of the centralized directory protocol. A more detailed description of the protocol may be found in [11].

The amount of memory required for the pointer is $\log N$ where N is the number of caches. The total amount of memory needed is thus $O(M \log N + Nc \log N)$ where M is the total size of main memory, N is the number of caches and c is the size of each cache. The above expression can be written as $O(M(1+k) \log N)$ where k is Nc/Nm (m being the amount of memory per node). We interpret k as the ratio of the size of cache memory per node to the size of main memory per node.

Assuming a constant value of k for the machine, the amount of memory required for the distributed directory scheme is $O(M \log N)$. We can expect then that, using the same technology, the cost of implementing the distributed directory scheme is significantly less than the fully mapped scheme—which requires $O(MN)$ amount of memory.

4 Performance Evaluation

We used two benchmarks to compare the performance of the fully mapped centralized directory protocol and the distributed directory protocol. The benchmarks consisted of an explicit partial differential equation solver (explicit PDE)[2] and a gaussian elimination program (gauss). These algorithms were chosen since they are widely used in scientific and engineering communities in applications requiring high performance computation.

Weak ordering was used in all the applications. For example, in the PDE algorithm used, for each element in the data array, two writes may be buffered at each time step before a fence [3] operation is required.

The simulation models were built upon an event driven simulation environment. The simulator uses traces that are generated "on the fly", in response to actual conditions at each instant in the simulated system, in order to preserve proper temporal ordering between the processors [12].

A mesh topology with 32-bit bidirectional channels was used for the comparisons. The caches were assumed to be 128 KB 2-way associative with a line size of 64 bytes. The SRAM cache to DRAM main memory access ratio was assumed to be 1:10. The directories for both the protocols was assumed to be implemented in SRAM whose cycle time was taken to be 1 cycle.

[2]The explicit solver used has data access patterns similar to those found in SOR and polynomially preconditioned conjugate gradient methods and so, while simple, is likely representative of a wider class.

BIBLIOTHEQUE DU CERIST

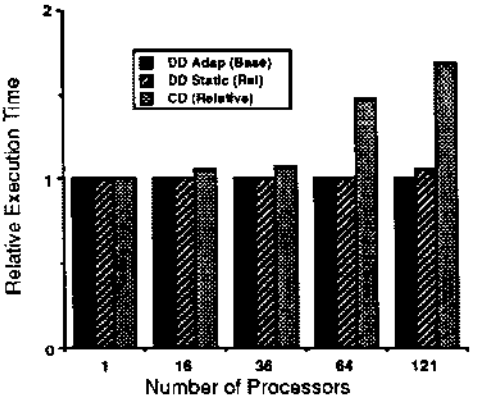

Figure 4: Explicit PDE with 1 network hop = 1 cycle

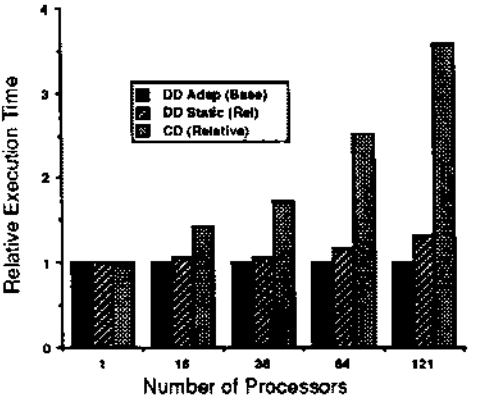

Figure 5: Explicit PDE with 1 network hop = 10 cycles

For one set of measurements, data was assumed to be transferred to a neighboring node (1 hop) in 1 cycle. This assumption would be true for systems using aggressive packaging techniques for the interconnection network. For another set of measurements, a slower network was assumed and data was assumed to be transferred to a neighboring node in 10 cycles.

Figures 4, 5, 6 and 7 compare the performance of the fully mapped centralized protocol and the distributed directory protocol (with and without adaptive routing) for the various cases. The execution time for the distributed directory protocol with adaptive routing was used as the base. The y-axis shows the relative execution time for the distributed directory protocol without adaptive routing and the centralized directory protocol as compared with the base. The x-axis shows the number of processors. The size of the input data set was kept constant.

For the explicit PDE solver the data was uniformly distributed among the nodes and the processes were randomly scheduled so as not to favor the distributed directory protocol. Figure 4 shows the relative execution time for a fast network which which requires one cycle for one hop. Figure 5 shows the relative execution time for a slow network which requires ten cycles for one hop. In the centralized directory protocol, the invalidations on a write can be done in parallel instead of sequentially as in the case of the distributed directory protocol. This can potentially cause the performance of the centralized directory protocol to be better if the number of caches that have to be invalidated is large, write buffering is ineffective, and the network is slow. However, for the explicit PDE benchmark, most of the communication is between two logical neighbors and the number of caches that have to be invalidated on a write is zero or one.

Figures 8 and 9 show a histogram for the number of readers between two successive writers for explicit PDE running on 64 and 121 processors respectively. The y-axis shows the percentage of times there were x number of readers between two successive writers,

BIBLIOTHEQUE DU CERIST

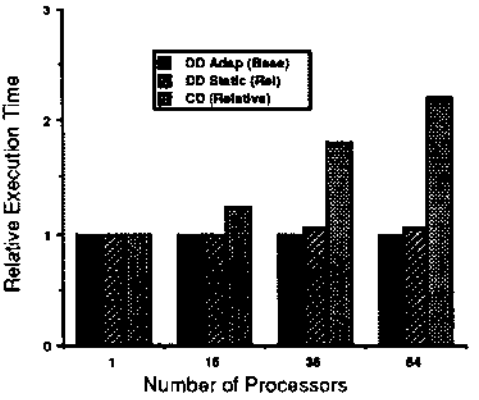

Figure 6: Gauss with 1 network hop = 1 cycle

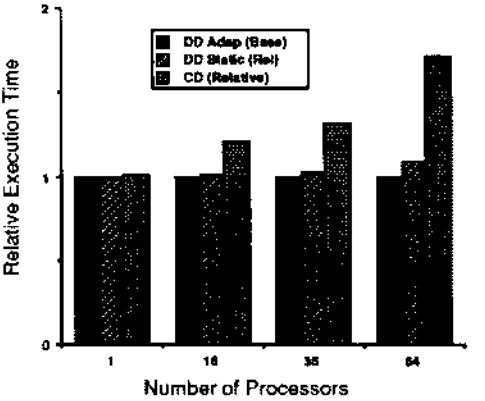

Figure 7: Gauss with 1 network hop = 10 cycles

where x is the value shown on the x-axis.

These measurements were done using the centralized directory protocol. For x equal 0, the data was either obtained from the main memory directly, or it was present in the requesting cache but the processor did not have permission to write to the data, in which case permission had to be obtained from the main memory by sending a modify-request signal and receiving a modify-granted signal before the write could be considered to be performed. For x equal 1, the data was present in one other cache, in which case that cache had to be invalidated and the data obtained from that cache in case it was dirty, before the requesting cache could be given permission to write to the line. For x equal 2 or more, the data was present in two or more caches which had to be invalidated.

As shown in figures 8 and 9, the data was present in at most one other cache most of the time for explicit PDE. In the distributed directory protocol, most of the requests require three or less network operations. On a write, if the requesting cache is the only cache that has a copy of the data, it also has permission to write to the line. Permission to write does not have to be obtained from the main memory in this case as in the centralized directory protocol. If one other cache has a copy of the data, the miss request is forwarded by the main memory to that cache which invalidates its copy and sends a reply directly to the requesting cache instead of sending it through the main memory (as in the case of the centralized directory protocol described in section 2). In the centralized directory protocol most of the requests require 4 or less network operations.

Figure 5 shows that the relative execution time of the centralized directory protocol became worse when the network was slowed down by a factor of ten. There was not enough opportunity for the centralized directory protocol to take advantage of the parallel invalidations since the data was not shared by many caches at the same time. A slower network results in more contention for the centralized directory protocol.

The advantage due to adaptive routing increases as the network becomes slower. This

BIBLIOTHEQUE DU CERIST

is shown in figures 4 and 5 by the difference in the relative execution time for the distributed directory protocol with and without adaptive routing. Techniques for adaptive routing [5] that are better than the one used for the simulations would further improve the performance of the distributed directory protocol.

Figures 6 and 7 show the relative execution times for gauss. Again, the distributed directory protocol performed better than the centralized directory protocol. For the gauss benchmark, figures 10 and 11 show the number of readers between writes for 36 and 64 processors respectively. The synchronization was done using an algorithm similar to a software barrier [9]. The degree of the tree structure used for the synchronization was 2. This accounts for the higher proportion of 2 readers between writes. For the gauss benchmark also, the length of the list of caches that had to be invalidated on a write was never large.

For the applications we have analyzed, the length of the list of caches that has to be invalidated on a write is small. This length depends more on the application than on the size of the system. This characteristic is also common to a range of applications studied in [13]. Thus, it seems that the distributed directory protocol would have good performance for a wide range of applications.

The distributed directory protocol has better performance since most of the requests can be serviced in three or less network operations verses four or less network operations in the case of the centralized directory protocol; the resource utilization is more distributed and there is no centralized bottleneck; and adaptive routing can be used to improve the performance in the case of congested networks. The direct cache to cache transfers used in the distributed directory protocol allows the performance to be more robust for more cost effective choices in main memory technology [10].

5 Conclusions

We have shown that the distributed directory protocol has good performance. The implementation of the distributed directory protocol is more scalable to larger systems than the centralized directory protocol. Simulation results have show that the distributed directory protocol has better performance than the centralized directory protocol. The protocol provides an efficient implementation of locks at minimal cost [11]. The scalability of the distributed directory protocol in terms of both cost and performance, makes it an attractive solution for the cache coherence problem in large scale systems.

References

[1] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An evaluation of directory schemes for cache coherence. In *Proceedings of the 15th International Symposium on Computer Architecture*, pages 281–289, 1988.

[2] James Archibald and Jean-Loup Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4(4):274–298, November 1986.

BIBLIOTHEQUE DU CERIST