A New Limited Directory Cache Coherence Scheme for Shared Memory Multiprocessors*

Phanindra K. Mannava, Akhilesh Kumar & Laxmi N. Bhuyan
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112.
E-mail: {mannava,akhil,bhuyan}@cs.tamu.edu
Phone: 409-845-5007

Abstract

The cache coherence scheme for a scalable distributed shared memory multiprocessor should be efficient in terms of memory overhead for maintaining the directories, as well as network latency for a memory request. In this paper, we propose a cache coherence scheme which minimizes the memory access delay and at the same time, reduces the directory overhead by using a limited directory scheme. In the proposed scheme, pointer overflow is handled by using a logically embedded ring for transmitting control messages. Wormhole routing, in conjunction with the usage of ring, introduces a snoopy behavior into the proposed scheme. We will show, with the help of several execution driven simulation results, that for real applications our technique outperforms the full map directory scheme, as well as the traditional implementations of limited directory schemes.

*This research has been supported in part by NSF grant MIP-9301959.

1 Introduction

Multiprocessors can be broadly classified into shared-memory and message-passing architectures. In message-passing multiprocessors, a node cannot access a memory location at another node directly. In order to access a remote memory location, a request is sent to the remote node, which then reads the memory location, embeds the data in a packet and sends it over to the requester. The programmer has to keep track of the data distribution and has to insert send/receive commands in the code anticipating a remote memory request. On the other hand, in shared-memory multiprocessors, memory addresses are mapped over all the nodes in the system and each processor can generate a memory reference assuming a unified view of the address space. Because of these reasons, shared-memory multiprocessors are easier to program and more popular.

Placing all the shared memory at one physical location would cause severe congestion due to all the memory requests being sent to this location. For this reason, distributing the shared memory seems to be a good idea. However, such an arrangement would require a fast communication technique for minimizing memory access latencies. Cache memory is used to keep a copy of recently accessed data to avoid frequent remote memory access. This requires that coherency be maintained among all the copies of a block in the cache and memory. The storage overhead to maintain the information about the states of blocks and its copies could be prohibitive for large number of processors. Also, the latency to service a cache miss could be very large for large systems for blocks that are widely shared.

Several cache coherence protocols have been proposed for implementing cache coherence efficiently. They can be classified into two categories: *snoopy* protocols and *directory-based* protocols. Because of their simplicity and efficiency, most of the shared memory multiprocessors have been built using snoopy protocols. The snoopy protocols need a broadcast medium, such as a single shared bus, such that the cache controllers can snoop on all shared memory transactions and take necessary action in their local caches. The broadcast medium in a snoopy protocol becomes the bottleneck, because of the limited bandwidth provided by a bus and also the limit on the number of processors that can be attached to a single bus.

A directory-based protocol avoids this problem by not requiring a broadcast medium. Several directory based cache coherence protocols have been proposed in the literature [1, 2, 3]. In all of these, each memory block is mapped to a home node which keeps a directory of the caches having a copy of this block. The directory entry for a block consists of the state of the block and a presence bit vector indicating the caches with a copy of the block. The scalability of such a system is limited by the size of the directory. This drawback can be overcome by designing limited directory schemes [4, 5, 3]. Limiting the directory size forces these schemes to resort to either broadcasting or limit the sharing of a memory block. It is observed that limiting the directory size could result in significant degradation in performance for some applications [6].

Recently some researchers have proposed snooping with point to point networks, based on the ring architecture [7, 8]. The addressing scheme is message based, however, as the message is sent around the ring, each node on the ring snoops on the message. Such a hybrid scheme exploits the good features of snoopy protocols in performing faster invalidations without being dependent on a shared bus. Multiple rings, connected by a higher level ring, are used for reducing the number of links traversed and network delay [8, 9]. As there is only a single path for all the packets, at high processor speeds or for applications having a high cache miss rate, the ring bandwidth becomes a bottleneck. Also, the latency of messages on the ring is usually larger due to the turn around delay on the ring.

In this paper, we propose a distributed shared memory architecture which combines the good features of the above mentioned schemes, while at the same time avoiding most of the drawbacks. The directory entries at the home node keep only a limited number of pointers to shared copies of a block. Our scheme uses limited directory and it is shown that performance will not deteriorate with increase in the number of processors and the degree of sharing in an application. These are achieved by using wormhole routing to pipeline messages over the network for smaller network delay, along with snooping to facilitate faster invalidations of shared blocks to maintain cache coherence in an efficient manner. When the sharing of a block exceeds the number of pointers in the directory, the proposed scheme uses an embedded ring over the interconnection network to provide a fast broadcast mechanism for snooping. The direct messages between a source-destination pair is sent over the shortest path provided by the network, whereas, the broadcast messages, such as invalidations, are sent over the embedded ring to be snooped by each node. This way, the messages are distributed over more paths and incur a smaller delay due to contention and the network does not get saturated.

The operation of our protocol is illustrated for a hypercube network with a ring embedded in it. This scheme can be extended to other direct networks, such as mesh, on which a ring can be embedded. We will denote our technique by Dir_iR, where R denotes a ring being used to handle the pointer overflow, and i denotes the number of pointers used. All the nodes snoop on the broadcast packets traversing the embedded ring. The rest of the messages use the regular hypercube routing. The communication switching is done by wormhole routing, thus hiding the message transmission latency to some extent.

In order to study the performance we have developed an exhaustive simulation model for the proposed architecture, including the cache coherence protocol and the network. This simulator is an extension of proteus, which is an execution driven simulator [10]. Four different application programs are used to evaluate the performance of the schemes under study. These are mp3d, matrix multiplication, Floyd-Warshall's all-pair-shortest-path algorithm, and FFT. The results indicate that the proposed scheme performs close to or even better than the full map directory scheme for all the applications.

The rest of the paper is organized in the following manner. The next section gives an overview of the related work. In the third section, we introduce our cache coherence protocol and its hardware requirements. The fourth section describes the simulation methodology, which also includes a description of each application and its sharing pattern. The next section includes the simulation results and their analysis. Finally we conclude the paper in section 6, by summarizing our findings and noting on further research.

2 Related Work

In this section we briefly review the cache coherence schemes for large-scale shared-memory systems. Several of these schemes attempt to address the issue of storage overhead due to directory to make the directory-based schemes scalable. Other schemes reviewed in this section attempt to develop an efficient coherence mechanism by supporting snooping over point-to-point networks.

In the full-map directory scheme [11], each node maintains a directory of all the memory blocks mapped onto that node. In this scheme, the directory entry for a block consists of the state of the block and a presence bit vector indicating the caches with a copy of the block. Although the full-map directory scheme performs well [6], the storage overhead is of the order of $O(B * P^2)$ which becomes very high with increase in shared memory size or number of processors. Here, B is the number of shared memory blocks per node and P is the number of processors in the system. In addition, if multiple messages have to be sent in response to a memory request, such as in the case of a write on a widely shared copy, number of messages to be sent and received is proportional to the number of copies in the system. Even though there might be non-overlapping paths between the source and different destination nodes, the messages have to be processed serially at the source. An example of this is a write on a read-shared copy where invalidations have to be sent individually to each node having a valid copy and the acknowledgments from these have to be processed serially at home once again.

Several limited directory schemes have been proposed in the literature to reduce the storage overhead [1, 4, 5]. They all have a common theme: limiting the number of entries in the directory. They differ in how directory overflow situation is handled. In one scheme [1], called $\mathrm{Dir}_i\mathrm{B}$ scheme, the pointer overflow is handled by adding a broadcast bit to the state information of each block. If the number of copies exceed the number of pointers, the broadcast bit is set and invalidations are done by broadcasting the invalidation message. In another scheme, called $\mathrm{Dir}_i\mathrm{NB}$ scheme, the pointer overflow is disallowed. If all the available entries in the directory are used, then the request for a new copy is satisfied by invalidating one of the copies to make room for the requesting node in the directory.

In the *superset scheme* [1], the directory is used to keep a composite pointer that can assume three states: 0, 1, and X — where X denotes either. When a new copy of the block is supplied, the bit pattern of the requesting node is compared to the existing entry. The existing entry may need to be modified such that it represents a superset including all the nodes with a copy of the block. When a write occurs to the block, an invalidation needs to be sent to all the nodes in the set represented by the composite pointer. The disadvantage with this scheme is the large number of invalidations which may be significantly larger than the number of copies [5].

In the *coarse vector scheme*, pointer overflow is handled by representing a fixed size region in the system through each bit in the directory [5]. In yet another limited directory scheme called LimitLESS, pointer overflow in the main directory is handled by allocating some space in the main memory to be used as extended directory entries [4]. But in all of the above cases either the number of valid copies in the system is limited or unnecessary invalidation messages are sent to nodes without a valid cache copies due to limited information at the home node.

Since our protocol uses a ring on a pointer overflow in the directory, we will survey some ring-based coherence schemes. These will reinforce the fact that a ring is better than bus for snooping. Barroso and Dubois [12, 7] use slotted ring with two types of slots, one for the request message and one for the data message. They show that for a small number of nodes (upto 40), processor utilization and network latencies are much better than in a shared-bus system. But their architecture is not scalable to a large number of processors since it is limited by the ring latency. Also, as the slots are predetermined, network utilization will vary with the program characteristics as some of the slots might be wasted depending on the ratio of request messages to data packets.

KSR-1 [8], too, uses the concept of a ring interconnection network. In this scheme, multiple rings are arranged in a hierarchical fashion. Each ring has 32 nodes, and 34 such rings can be connected together. It uses slotted rings, where a slot is of the size of a data packet. Even though it is a cache-only memory architecture, the main idea is to limit the invalidation and search messages from traversing to all the nodes. Here the slot size is very large, and for a request packet most of the slot space is wasted. Farkas, et al. [9] also have proposed a coherence protocol for a hierarchical ring based multiprocessor. They also use slotted ring.

3 The New Limited Directory Scheme

As mentioned earlier, the proposed cache coherence scheme is a limited directory scheme. A directory at the memory is kept with a number of pointers limited to much less than that in the full map. The identity of the node requesting a read copy is entered into this directory and when the directory becomes full a broadcast bit is set. Setting the broadcast bit indicates that there is no more pointer space in the directory and a snooping ring will control the operation. When an invalidation request is received for a read-shared block, the entries in the directory is checked. If the broadcast bit is not set, individual invalidation requests are sent to each node with a copy, and acknowledgments are expected from all these nodes, before the write may proceed. In case the broadcast bit is set, a single invalidation message is sent over the network along a logically embedded ring. The source as well as the destination node for this message are the home node itself. Each node in the path snoops on this message and takes appropriate action in its cache.

A cache block at a node can be in one of the three states: *Invalid*, *Read-Shared*, or *Write-Exclusive*. These states indicate whether the block is not current in the cache, present in read-only mode, or present in read-write mode, respectively. Similarly the blocks in memory can be either in *Valid*, *Not-Cached* or *Dirty* state. A valid memory block indicates that the data in the memory is up-to-date and there could be read-shared copies of this block in multiple caches. A dirty memory block indicates that the data in the memory is stale and one of the caches has a write-exclusive copy of the block. A not-cached state indicates that the copy in the memory is the latest and the only copy.

Message Type	Request Type	Condition
Cache to Home	Read/Write/Invalidation requests Acknowledgments of receipt of requested data Acknowledging inval of a read-shared copy Replacements and directory updates	On a cache miss
Home to Cache	Requested memory block Ack for completion of invalidations	
Cache to Cache	Forwarding cache block	From write-exclusive copy to requester (Dir_0R case)
Ring Messages	Invalidation message Search/SearchI for a write-exclusive copy	No of Shared copies $> i$ for Dir_0R case

TABLE 1: Message Categorization

The messages sent in this scheme are categorized as cache to home node, home node to cache, one cache to another, and ring messages (as shown in Table 1). The first three of these use the usual shortest path routing from between source to destination. The ring messages as the name implies use the embedded ring. Some of the cache to home node messages are generated due to a cache miss, such as read request to an absent or an Invalid cache line, write request to an absent, Invalid or Read-Shared cache line. The last of these results in an invalidation request being sent to the home node.

The response of home node to a request depends on the type of request as well as the state of the requested block in the memory. For a read request on Valid or not-cached block, the home node sends the requested block to the requester immediately and updates its directory. If the directory pointers are full then the broadcast bit is set. If the broadcast bit is already set indicating a directory pointer overflow, no action in the directory entries is required. On a write request to a block which is not-cached or Valid, a copy is sent to the requester. Also, in the case of Valid state, an invalidation message is sent to all the caches with a copy. For an invalidation request also, all

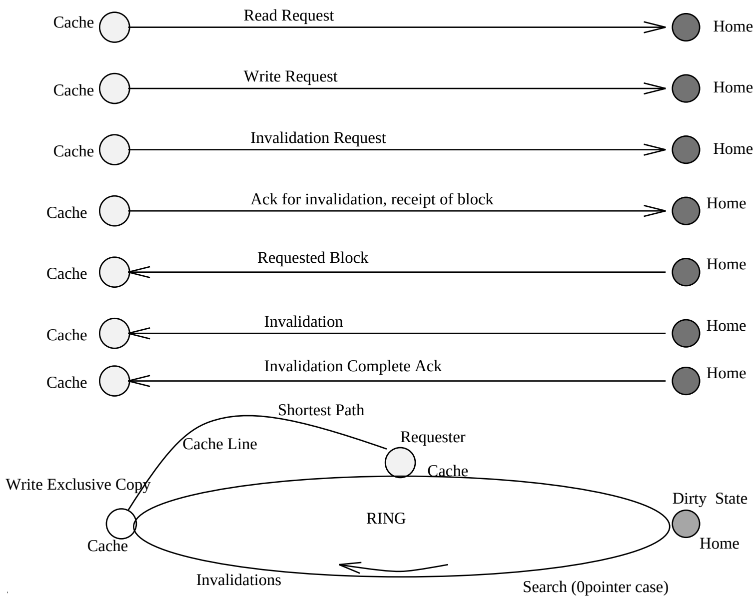

Figure 1: Different Message Types

the read-shared copies are invalidated and after receiving all the acknowledgments an invalidation acknowledgment is sent to the requester. As mentioned earlier, if the number of copies is more than the number of entries in the directory, the invalidation is sent over the ring. Otherwise, individual invalidation requests are sent to each node holding a valid copy. This memory access is complete only when all the invalidation requests are acknowledged or the ring message comes back to home and the requesting node acknowledges the receipt of a copy.

In case of zero pointers, search messages are sent on the ring searching for the node with the write-exclusive copy. Upon receipt of such a message, the cache with the write exclusive copy supplies a copy to the requesting node and changes its own status to invalid or read shared depending on whether the request was a read request or a write request. The requesting node, upon receiving the copy of the block, forwards a copy to the home node to update the memory. The block is supplied to the requesting node via the shortest path in the network.

Replacements of a cache copy also causes sending of an update message to the home node. If the replaced cache block is in write-exclusive state, a copy of the cache line is also sent along with the update message. If a request is received for a block at home, for which another request is still in progress, the new request is placed in a queue. Once the home node completes processing a request, it picks up the next request from the head of the wait queue. These messages are shown in figure 1. The labels on the arc are the messages being sent from one node another.

One obvious advantage of our scheme is in terms of the number of messages to be processed at the home node and the number of links used for a memory access to be completed. Even though the network is very fast, processing delay is directly proportional to the number of messages sent and received. Also, the number of links connected to a node are limited. Therefore, a fraction of the messages have to be sent out and received serially. In the case of an n-dimensional hypercube, there are n links to each node and e-cube routing causes half the number of messages sent or received at a node to use the same link. In the proposed scheme, we reduce both the number of messages sent out and the number of links used.

3.1 Hardware Design

A hardware design for the proposed protocol will depend on the characteristics of the network. As examples, figure 2 shows hypercube and mesh network with embedded rings in dark lines. The figure also shows the details of hardware blocks inside a node for different communications. For example, the routing of the messages in a hypercube is done by two techniques: one called e-cube routing for direct messages and the second ring routing for ring-messages. In the following we will consider hypercube as the underlying network and explain the hardware design accordingly.

The e-cube routing is the most commonly used routing scheme in hypercubes [13, 14] because of its simplicity and ease of implementation. In this scheme, the node receiving the message compares its own address with the address of the destination node of the message. If the addresses differ in the i-th bit position, the message is required to be forwarded on the i-th dimensional link. In e-cube routing, a message is always forwarded on the link along the lowest dimension among the required dimensions.

The logical embedding of a ring in a hypercube is done using a binary coding scheme called *reflected gray code* [15]. A simple gray code is a sequence of binary numbers in which the hamming distance between adjacent numbers is one. This guarantees that in a hypercube the adjacent numbers have a link connecting them. A special type of gray code, in which the last number and the first number differ in a single bit location too, is called a reflected gray code.

In order to decrease the network delay, we are using wormhole routing for both e-cube as well as ring routings. Wormhole routing is very common in the latest mul-

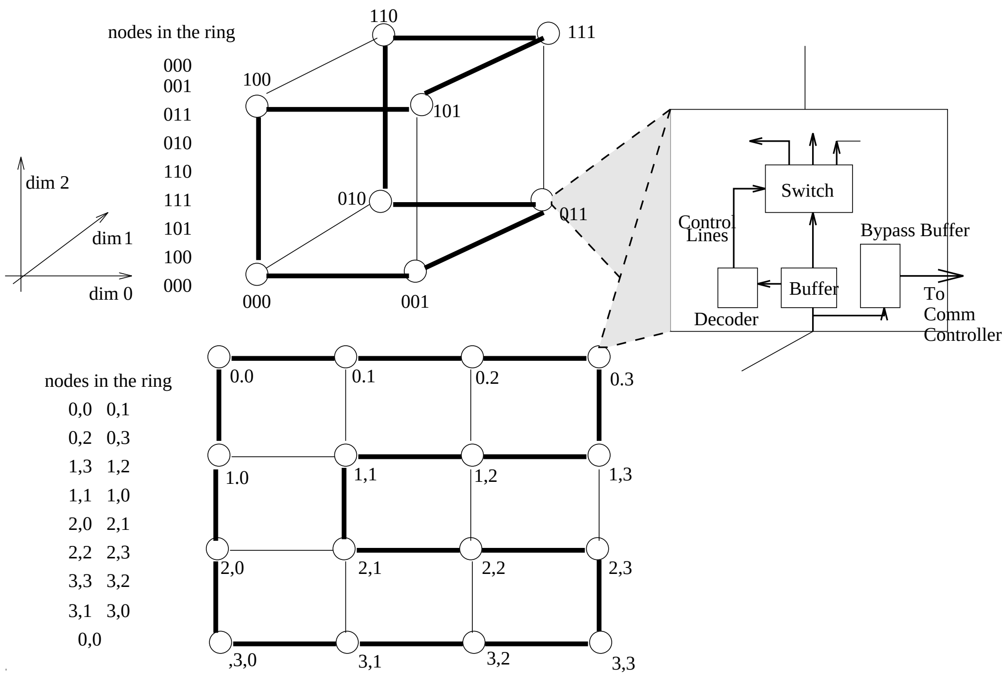

Figure 2: System Configuration

tiprocessor systems [13, 14]. A message, in wormhole routing, is made up of flits, and moves in a pipeline fashion from the source node to the destination node in a pipeline manner. This reduces the delay considerably in transmitting a message from source to destination when the interference due to other messages in the network is low. The speed is achieved by forwarding the message as soon as the header enters a node and is decoded. The decoding enables the node to properly switch the header to the required outgoing port. Wormhole routing has been used for e-cube messages before, but using wormhole routing for ring messages introduces some new features to our design.

The first one is that the ring latency can be reduced considerably. Also, snooping can be done by copying the flits as they pass through a switch. This is done by having an extra buffer (bypass buffer in Figure 2) in the switch, and a multiplexor which forwards the incoming message to both the bypass buffer as well as storage buffer. Once the whole message is copied into the bypass buffer, it can be processed and the cache directory can be updated. In the case when the received message is a invalidation message and the cache has a read-shared copy, it is invalidated. In case of Dir_0R search messages, searching for a write-exclusive copy also traverses this ring. If the node with write-exclusive copy receives a SEARCH message, it forwards a copy of the cache block to the requester and changes its own state to read-shared. If the message is SEARCHI, a copy is forwarded to the requester and its own copy is invalidated.

E-cube routing by itself guarantees a deadlock free communication. When e-cube is combined with the ring routing, a situation might occur when there is a circular wait for buffer space on the ring. This can be avoided by increasing the number of buffers at each switch. If we assume that there can be only one outstanding request from each processor at a time, the number of ring messages in the network at any time is bounded by $O(N)$. Since the size of each messages is limited, 8-10 bytes for control messages and 40-100 bytes for messages containing a memory block, the total buffer space requirement is limited.

Another method of avoiding deadlocks is by using virtual channels. In our method by using just three channels we can avoid deadlocks. Two channels are assigned for ring messages and the third one for e-cube routed messages. We will refer to these as VC_0, VC_1 and VC_2. All ring messages travel along VC_0, unless they get blocked in which case if the blocking message has lower node number as the source then they move up to VC_1. Once a message moves up to VC_1 it will never move back to VC_0. All blocked messages get stored completely at a the blocked node. In such a situation we see that ring messages from node 0 will never move up to VC_1 as they have the lowest priority in the network will never be blocked by a lower priority message. Thus there are N number of VC_1 buffers and only a maximum of $N-1$ messages competing for them. This avoids any circular wait. Also as all the blocked messages have to stored completely. Each virtual channel has a control packet size buffer space at each node.

The routing of e-cube routed messages is independent of the ring routed messages. These use VC_2 and because e-cube routing guarantees a deadlock free routing, they require only a single flit buffer space at each node. By using the above scheme we are able to avoid deadlocks by using just a maximum of 3 buffers for each link.

4 Simulation Methodology

4.1 Application Programs

We have run extensive execution driven simulations for various application programs. For this purpose an existing simulator, called proteus [10], has been extended to include our network and cache coherence models. The application programs used are MP3D from the SPLASH benchmark suite. FFT, matrix multiplication, and Floyd-Warshall were developed in-house.

MP3D is a three-dimensional particle simulator used in rarefied fluid flow simulation. We used 3000 molecules and ran the simulation for 10 time steps to keep the simulation time manageable. For Fast Fourier Transform (FFT), we used a version of Cooley-Tukey FFT optimized for cache based shared memory system. The simulations are done on an input of 2^{14} points. Floyd-Warshall's algorithm determines all-pair-shortest-paths among the nodes in a graph. The graph used for the simulation has 128 nodes and the initial weights were randomly assigned to the edges. For matrix multiplication two 128x128 matrices were multiplied.

To interpret the performance of various coherence schemes, it is important to understand the sharing pattern exhibited by shared blocks in an application. Therefore, we show a typical sharing behavior for the four applications for 32 processor system. The sharing pattern is obtained by determining the number of copies of a block before a write miss is serviced by the home node. These are plotted for a full-map directory case on a semi-logarithmic scale where horizontal axis shows the number of shared copies and vertical axis is number of occurrences of each. We see that in figure 3, FFT sharing is very low, most of the data is not shared at all and a small number is shared by one or two processors. On the other hand,figure 4 shows that the sharing is distributed over the whole range in Floyd-Warshall. It is still high at the low end and again moderate in the middle and tapering off at other places. This is due to a shared variable being used in its outer loop which is accessed by all the processors and updated in every iteration.

In MP3D, too, most of the writes are to the blocks that are shared among few processors as shown in figure 5. However, this application also shows that there are quite a few writes to widely shared blocks. This is because the algorithm uses several locks and barriers to order the steps. These locks and synchronization variables are shared among all the processors. As seen in figure 6, matrix multiplication shows almost no sharing for shared writable blocks. In this implementation, the output matrix is divided into rectangular blocks and one block is assigned to each processor. The processors share the rows and columns of the input matrices but there are no writes to it, so, it does not generate any invalidations.

4.2 Simulation Parameters

The simulation parameters for our measurements are shown in Table 2. Main memory access time is 4 times slower than cache access time. The channel width is equal to

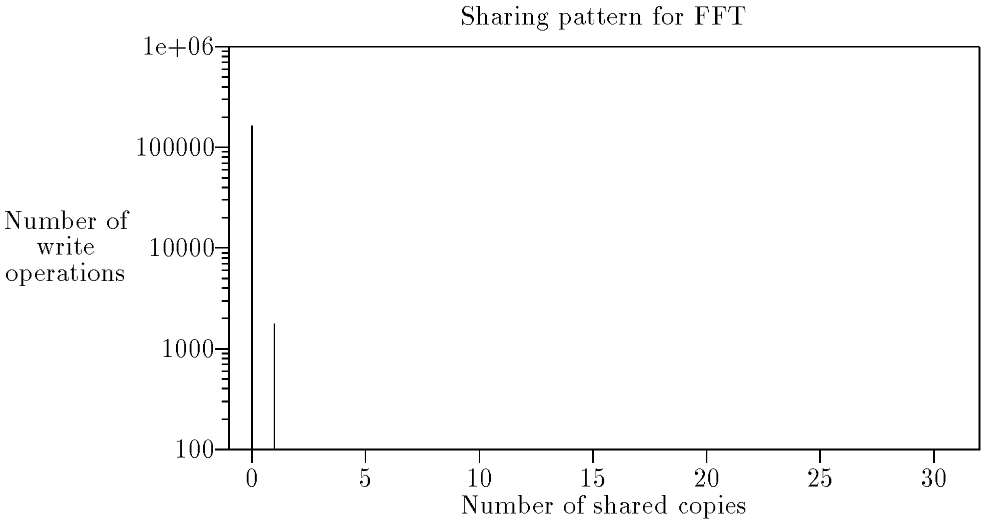

Figure 3: Sharing Pattern for FFT for 32 Processors.

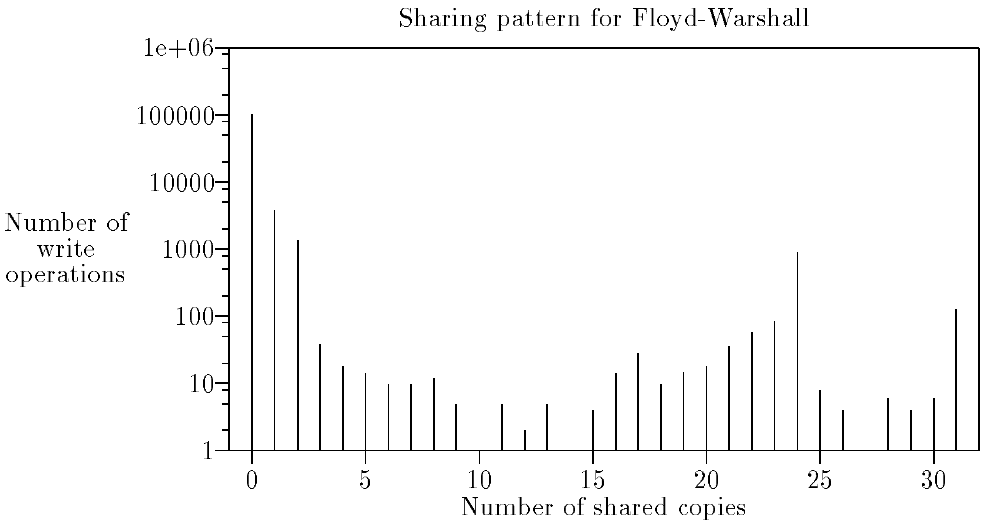

Figure 4: Sharing Pattern for Floyd-Warshall for 32 Processors.

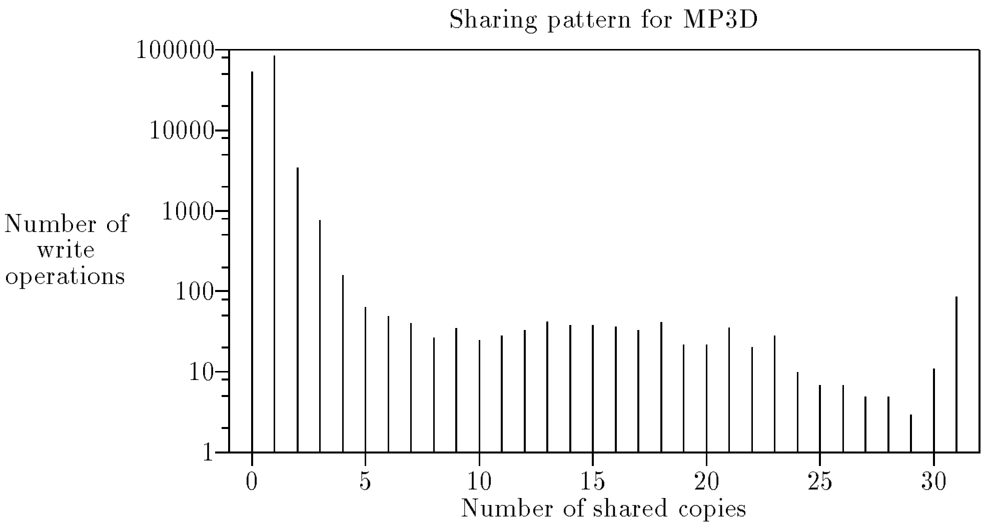

Figure 5: Sharing Pattern for MP3D for 32 Processors.

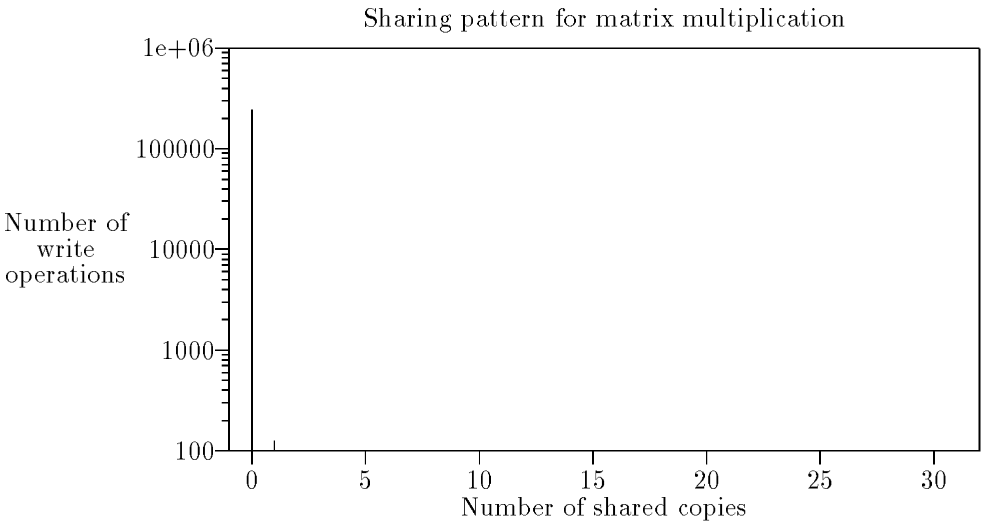

Figure 6: Sharing Pattern for matrix multiplication for 32 Processors.

flit length, which means a flit can traverse a link in one processor cycle. The header decode and switching time is 1 cycle.

Parameter	Value (in processor cycles)
Processor Cycle	1
Cache Access Time	1
Memory Access Time	4
Wire Delay	1
Node Switching Delay	1
Channel Width	16 bits
Flit Length	2 bytes
Line Size	32 bytes
Set Size	1
Request packet length	8 bytes
Data packet length	40 bytes

TABLE 2: Simulation Parameters

A packet is sent out on a link only when it becomes free. Also only one message is sent to the network from a network interface in one cycle. Similarly, only one message can be processed by the network interface in a cycle.

The two types of packets in the system are, the control packet and the data packet. The control packets carry the source node number, destination node number, packet type and request type only. This can be done in 8 bytes, thus making the control packet length 8 bytes long [16]. The data packet also includes a memory block thus its length is assumed to be (Line Size + 8).

5 Results and Discussion

In this section, we present the results of the simulation for the proposed scheme and compare it with the simulations of full-map and limited directory schemes. The proposed limited directory scheme with embedded ring is identified by Dir_iR, where i denotes the number of pointers in the limited directory. The Dir_0R scheme is a special case without any pointers in the directory, and is somewhat similar to the ring architectures suggested in [7, 12]. However, in the proposed Dir_0R scheme, only some of the messages are sent over the ring, the rest other messages go to the destination through one of the shortest path. Therefore, the performance of our Dir_0R scheme is actually better than what one would expect from [7, 12]. As we will see later, Dir_0R performs worst in most of the cases.

We also simulated our Dir_iR scheme for various number of pointers. The full-map scheme is simulated by having as many number of pointers as the number of processors. Dir_iB scheme [1] is also simulated for 2 pointers. The pointer overflow in Dir_2B is handled by setting a broadcast bit and by sending invalidation messages to every node on a write request. One could broadcast a message in $O(\log P)$ steps in a hypercube, but this requires involvement of other nodes during the intermediate steps of the broadcast which is usually slow due to message processing time at these nodes. Gathering of acknowledgments after a broadcast of invalidations also requires intervention by intermediate nodes. In our evaluations, we chose to send individual messages to each of the nodes in the system rather than using any special broadcast algorithm.

5.1 Message Processing Overhead

One major advantage in the proposed scheme is that the number of messages sent and received for invalidating widely shared data is reduced. Even the number of links traversed is also reduced. This is illustrated in Table 3 for a 5-dimensional hypercube. In this table we list the number of messages sent and received and number of links used for different number of shared copies. A node sends an invalidation request to home only if it also holds a read-only copy of the block, thus the number of invalidations sent from home in response to the request is one less then the number of shared copies in the system.

We have listed full map, our limited directory scheme Dir_1R, Dir_2R, Dir_0R and the traditional limited directory implementation with two pointers Dir_2B. We clearly see that for the case when the number of shared copies is medium to large, our scheme uses much less number of links and for all cases beyond the number of pointers, our scheme just sends out and receives a single invalidation message over the ring.

Protocol	Shared Copies	Mess Sent	Mess Recvd	links used
Full Map	1	0	0	0
Dir_0R	1	0	0	0
Dir_1R	1	0	0	0
Dir_2R	1	0	0	0
Dir_2B	1	0	0	0
Full Map	2	1	1	5
Dir_0R	2	1	1	32
Dir_1R	2	1	1	32
Dir_2R	2	1	1	5
Dir_2B	2	1	1	5
Full Map	16	15	15	75
Dir_0R	16	1	1	32
Dir_1R	16	1	1	32
Dir_2R	16	1	1	32
Dir_2B	16	31	31	155
Full Map	32	31	31	155
Dir_0R	32	1	1	32
Dir_1R	32	1	1	32
Dir_2R	32	1	1	32
Dir_2B	32	31	31	155

TABLE 3: Activity at home node due to Invalidation Request from a cache with a Read Shared copy

The link usage in the above was calculated with the assumption that in a 5 dimensional hypercube the average distance a message travels is 5/2, which is 2.5 links. For each invalidation message sent out, an acknowledgment is expected which means that each invalidation uses 5 links. For example if the number of shared copies is 16, for a full map case 15 invalidations have to be sent out (recall that no invalidation message needs to be sent to the requesting node). This means $15 \times 5 = 75$ links are used. On the other hand for a ring based scheme the invalidation message is sent over the ring, which uses only 32 links. In the extreme case of Dir_2B, invalidation messages have to be sent to all the 31 nodes.

5.2 Simulation with Synthetic Trace

Initially, we ran a synthetic trace driven simulation to study the behavior of our protocol. The write times for a 32 processor hypercube for full map, 0 pointer (Dir_0R), 2 pointer (Dir_2R) and 5 pointer (Dir_5R) schemes are shown in Figure 7. It shows that for low number of invalidations, all methods, except 0 pointer, have almost same write miss service delay. The 0 pointer case has higher delay because even the reads to dirty copies have to send a message on the ring searching for the write-exclusive copy. As the number of invalidations is increases, the service time for full-map scheme increases linearly, whereas the delay levels off for others cases. This is because once the directory pointer overflow occurs all invalidation message traverse the ring, which has a constant latency.

Also, the crossover point between the proposed scheme and full-map scheme depends on the communication speed. As the communication speed increases over the processor speed, the crossover point moves towards left.

A ring message traversal time can be calculated to be

$$\text{T} = \text{packet setup time} + (\text{switching time} + \text{link delay})\text{xP} + L_p\text{-1}$$

Where P is the number of processors in the system and L_p is the length of a packet. If all the delays are assumed to be just one cycle and length of the packet to be one flit, the total ring traversal time for a 32 processor system with wormhole routing is just 68 cycles. When this message is received back, only one message processing is tacked on to the ring traversal time.

In full-map scheme, where 15 invalidations have to be sent out, 15 packet set up times are involved and e-cube routing features are such that half of these will go out over the same link. So, the last message will go out after 15 cycles assuming that it is not blocked by any earlier sent message. A message travels on 2.5 links on the average to reach the destination, which takes 5.5 cycles. The packet has to be received and processed at the receiving node by its cache controller which sends back an acknowledgment. This acknowledgment has to be set up and sent back to the home node. So, in this case, in addition to the network transmission delay, the packet processing time at the destination and 15 packets' send-process time and 15 acknowledgment packets' receive-process time at the home node is also involved. Our scheme clearly has less processing overhead.

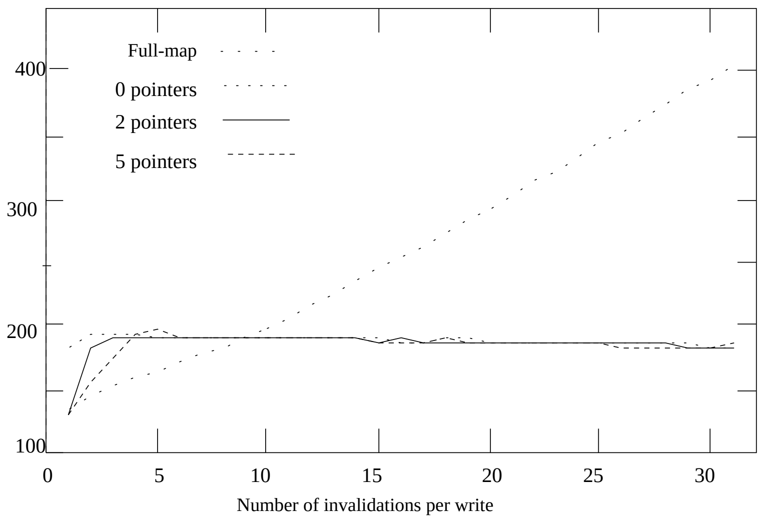

Figure 7: Invalidation Times for a 5 dimensional hypercube

5.3 Execution Driven Simulation

We ran execution driven simulations for the four applications described earlier, for 2, 3, 4 and 5 dimensional hypercubes. The schemes compared in these simulations are full map, Dir_0R, Dir_1R, Dir_2R and Dir_2B. The plots are for execution time on the vertical axis and different schemes and system sizes on the horizontal axis. The execution time is plotted as percentage of the execution time for full map. This is done because among all the directory schemes full map is known to give the minimum execution time.

The results for MP3D are shown in Figure 8. In this application, for a small system size of 4 or 8 processors Dir_1R, Dir_2R behaves similar to full-map. As the system size is increased to 16 and 32 processors, Dir_2R performs better than the full-map, whereas Dir_1R performance is as good as the full-map. This performance is in accordance with the sharing pattern characteristics of this application. As the sharing is high at the low and at the high end, for the sparsely shared blocks, the write time is similar as for Dir_2B and full-map. But for widely shared blocks, which is beyond performance crossover point of figure 7, Dir_2R scheme write incurs less delays than full-map.

On the other hand, Dir_0R performs worse than the full-map scheme. This is because of large number of messages going over the ring that saturates it. We have also compared our limited directory scheme with the traditional implementation of limited directory Dir_2B and see that its performance is worse than the full map, Dir_1R and Dir_2R schemes. The performance of Dir_2B scheme deteriorates with increasing

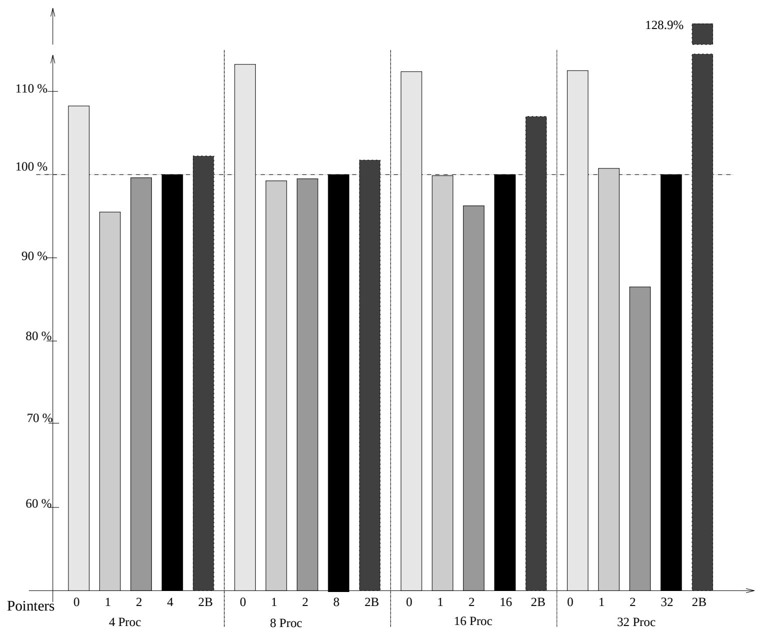

Figure 8: Simulation results for mp3d.

number of processors because of the large number of messages required to be sent due to broadcasting.

In FFT, for 4 and 8 processor systems, all the schemes except Dir_0R perform almost same as shown in Figure 9. Dir_0R performance is the worst and with the increase in the system size it deteriorates even further. As the size increases to 16 and 32 processors Dir_1R also shows a worsening trend, but Dir_2R, full map and Dir_2B perform very close to each other. This was expected from the sharing pattern. For FFT, the block is either shared by 0 or 1 processors. This means that when a write request comes for read-shared copy the home node has to send atmost one invalidation to some other node. In such a case, Dir_2R and Dir_2B and full map would need to send same number of invalidation copies. The behavior of Dir_1R can be explained by showing that when an invalidation request is sent to home, at most one other copy of the requested block exists, making it a total of two pointers needed to point to the copies (one is the requester itself). This causes a pointer overflow in Dir_1R and the invalidation message has to be sent on the ring.

In Floyd-Warshall, there is not much difference between the various methods for small number of processors. As the number of processors increases, the Dir_2R execution time decreases compared to the full-map scheme, as shown in 10. Again, the performance of Dir_0R worsens with increase in the size of the system but the deterio-

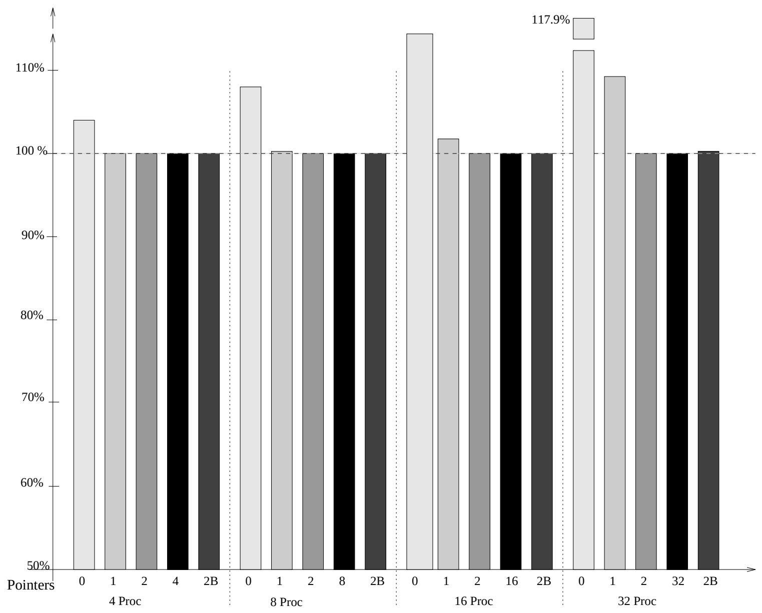

Figure 9: Simulation results for FFT.

ration is not as bad as for previous two applications. This is due to the sharing being distributed over the whole range, thus compensating the advantages of one method over the other by equally distributing the sharing to both sides of the performance crossover point.

The sharing pattern of matrix multiplication is an extreme case. Most of the blocks are only read shared and very little write sharing is observed due to false sharing. This means every cache miss results in a read request to the home and the home always has a clean copy to supply to the requester. If a write request is sent to home, then it is usually not cached by any other cache. Again this is reflected in the performance results shown in Figure 11. All the schemes perform equally well in this case.

6 Conclusion

In this paper we have proposed a new limited directory cache coherence scheme. In addition to decreasing the directory overhead, the scheme also reduces the network latencies. The communication time is reduced by using wormhole routing and by implementing the broadcast caused by pointer overflow in a more efficient manner over a logically embedded ring. This ring uses wormhole routing and provides a snooping capability to a limited directory scheme. The proposed scheme can be realized on any

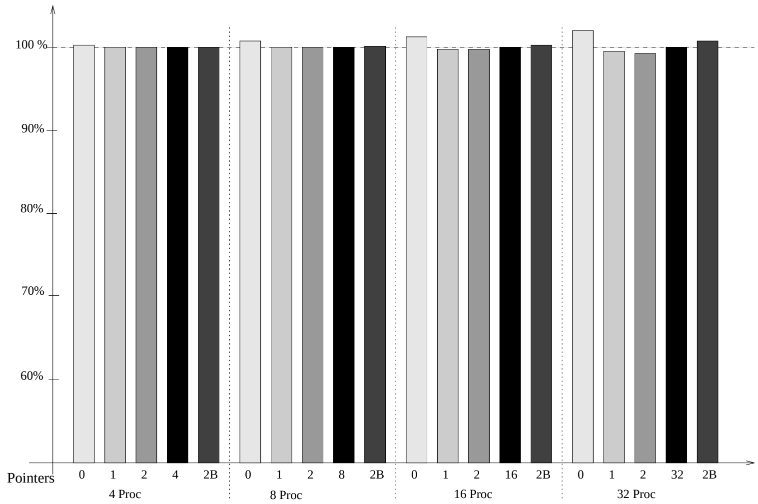

Figure 10: Simulation results for Floyd-Warshall all-pair-shortest-path algorithm.

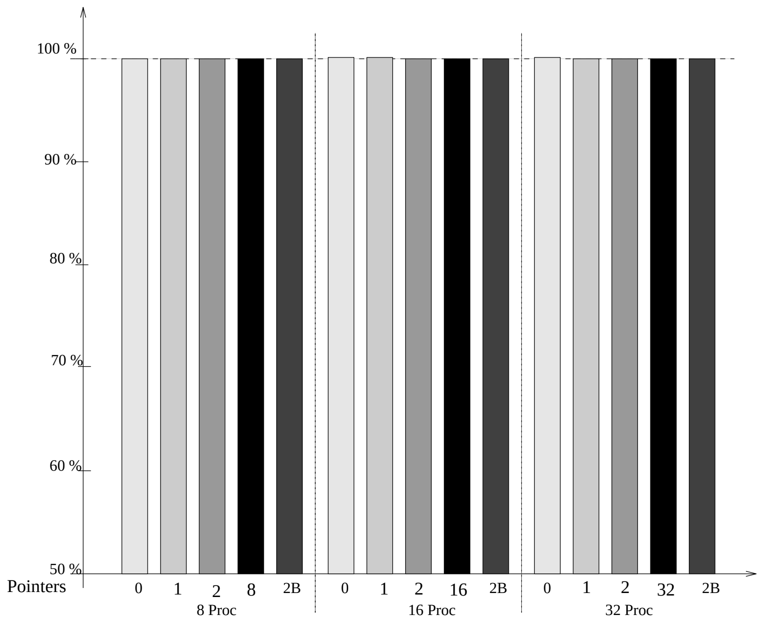

Figure 11: Simulation results for matrix multiplication.

direct network as long as a ring can be embedded on to it.

Exhaustive simulations have shown that for different types of sharing patterns, from no sharing in matrix multiplication to widely distributed sharing in MP3D and sharing at the low and high end in Floyd-Warshall, the proposed scheme works as well as the full-map scheme, and in some cases even better. This observation shows that the directory overhead can be reduced without losing performance. The simulations also show that 2 pointers are sufficient for most of the applications. It would seem that for larger number of processors, the ring latency would overshadow any of the benefits gained by using our technique. While we agree with this observation, our preliminary results for 64 processor system show that for MP3D our Dir_2R scheme still works better than the full-map scheme.

We can also improve the performance of our scheme by using multiple edge disjoint rings in a hypercube. It is known that $n/2$ edge-disjoint rings can be embedded in a n-dimensional hypercube [17]. Such a scheme would distribute the network traffic evenly over all the links, thus decreasing contention in the network.

References

[1] A. Agarwal, R. Simoni, J. Hennesy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," *Proc. 15th Annual International Symposium on Computer Architecture*, pp. 280–289, 1988.

[2] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," *IEEE Computer*, vol. 23, no. 6, pp. 12–24, June 1990.

[3] D. J. Lilja, "Cache coherence: Issues and comparisons," *ACM Computing Surveys*, vol. 25, no. 3, pp. 303–338, September 1993.

[4] D.Chaiken, J. Kubiatowicz, and A. Agarwal, "Limitless directories: A scalable cache coherence scheme," In *Proceedings of ASPLOS-IV*, pp. 224–234, April 1991.

[5] A. Gupta, W.D.Weber, and T.Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," In *Proceedings of International Conference on Parallel Processing*, volume I, pp. 312–321, 1990.

[6] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *IEEE Computer*, vol. 23, no. 6, pp. 49–58, June 1990.

[7] L. A. Barroso and M. Dubois, "Cache coherence on a slotted ring," In *Proceedings ofInternational Conference on Parallel Processing*, pp. 230–237, August 1991.

[8] "KSR1 Technical Summary," Technical report, Kendall Sqare Research Corp, 1992.

[9] K. Farkas, Z. Vranesic, and M. Stumm, "Scalable cache consistency for hierrarchically- structured multiprocessors," In *Proceedings of Supercomputing*, pp. 348–357, 1992.

[10] E. A. Brewer and C. N. Dellarocas, *Proteus: User Documentation , Version 0.5*, MIT, Cambridge, MA, 1992.

[11] L. M. Censier and P. Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Trans. Comput.*, vol. C-27, no. 12, pp. 1112–1118, Dec. 1978.

[12] L. A. Barroso and M. Dubois, "The performance of cache coherent ring based multiprocessors," In *Proceedings of International Symposium on Computer Architecture*, pp. 268–277, May 1993.

[13] "nCUBE/2 processor manual," *nCUBE Corp.*, 1990.

[14] "Intel ipsc/2," *Intel Corporation*, 1988.

[15] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[16] A. Kumar, P.K.Mannava, and L.N.Bhuyan, "Efficient and scalable cache coherence schemes for shared memory hypercube," In *Proceedings of Supercomputing*, pp. 498–507, 1994.

[17] S. L. Johnsson and C.T.Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, pp. 1249–1268, 1989.