Efficient Implementation of Cache Coherence in Scalable 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 an efficient invalidation mechanism using logically embedded rings. rings for transmitting control messages. A single ring architecture for small scale multiprocessor and a multiple ring hierarchical architecture for a scalable multiprocessor are evaluated. In both the architectures, wormhole routing, in conjunction with the usage of ring, introduces a snoopy behavior to the proposed scheme. We will show, with the help of execution driven simulation results, that for several applications our techniques outperform the full map directory scheme, as well as the traditional implementations of limited directory schemes.

Keywords: cache coherence, shared memory multiprocessor, hierarchical rings, hypercube, snooping

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

1 Introduction

Shared-memory multiprocessors are easier to program and are 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 caches 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 and 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 [1, 2]. 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 due to the limited 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 [3, 4, 5, 6]. 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 limited directory schemes. Limiting the directory size forces these schemes to resort to either broadcasting or to limit the sharing of a memory block. It is observed that limiting the directory size results in significant degradation in performance for some applications [7].

Recently some researchers have proposed snooping with point to point networks, based on the ring architecture [8, 9]. 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 [9, 10]. 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 avoids most of the drawbacks. The directory entries at the home node keep only a limited number of pointers to shared copies of a block. It will be shown that, unlike most other limited directory schemes, performance does not deteriorate with increase in the number of processors and the degree of sharing in an application. The advantages 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 embedded rings over the interconnection network to provide a fast broadcast mechanism for snooping. The direct messages between a source-destination pair are sent over the shortest path provided by the network, whereas, the broadcast messages, such as invalidations, are sent over the embedded rings 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 single or multiple rings embedded in it. This scheme can be extended to other direct networks, such as mesh, in which a ring can be embedded. When multiple rings are used they are arranged in a two level hierarchy. Multiple lower level rings (sub rings) are connected using a single upper level ring (super ring). 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 [11]. Five different application programs are used to evaluate the performance of the schemes under study. These are MP3D and WATER from SPLASH benchmarks [12], and matrix multiplication, Floyd-Warshall's all-pair-shortest-path algorithm and FFT developed by us. 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 system organization. In the third section, we introduce our cache coherence protocol. The fourth section describes the simulation methodology, which also includes a description of each application and its sharing pattern. The fifth 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 System Organization

2.1 Hardware Organization

In the multiprocessor organization being considered, nodes are connected together using a point to point network, specifically a hypercube. Each node has a processor, a portion of the shared memory and a communication controller. The communication controller interfaces the node switch to the processor. The switch has $log_2 N$ input and output links to other switches and one input/output link to the local processor. As soon as the header flit of a message enters a switch it is decoded and forwarded to the appropriate output link. If the destination is the local processor it is forwarded on the local link.

A logical ring is embedded in the hypercube using reflected Gray code (an example is shown in Figure 1) [13]. To embed multiple rings we divide the hypercube into subcubes and embed a sub ring in each of the subcubes. A different ring (denoted as super ring) is embedded connecting a single node from each of these sub rings. In Figure 3 we show a five dimensional hypercube with four sub rings. First the five dimensional hypercube is divided into four subcubes 00XXX, 01XXX, 11XXX and 10XXX. One representative node from each is found by replacing XXX by 000, (XXX could be replaced by any of the other seven choices such as 001, 010 etc). Then 00000, 01000, 11000 and 10000 form the super ring. The set of links in the sub rings as well as the super ring are disjoint. This will be shown to be important later as it distributes the network traffic over larger number of links.

2.2 Multicasting and Broadcasting in a Cache Coherence Environment

In a typical invalidate cache coherence protocol, a write by a processor to a read only copy generates an invalidation request to home. The home needs to send invalidation messages to all the other caches with a valid copy of the requested block. In a snoopy scheme, a single broadcast message is sent on the bus, and all the snoopers detect this invalidation message, check their caches and if a copy is found its state is changed to invalid. In a network which is not bus based, such as mesh, hypercube, MIN etc, a multicast or broadcast has to be done to transmit the invalidations to all the copies. If the identity of the caches with a copy is known, as is the case in full map, individual invalidation messages are sent to each one and acknowledgments are expected from each also. In the case of limited directory schemes in which some times identity of all the caches with a copy is not known, a broadcast has to be done.

In a hypercube, e-Cube routing has the property that on the average half the messages transmitted from a node go out on the lowest dimension link. Thus if multiple

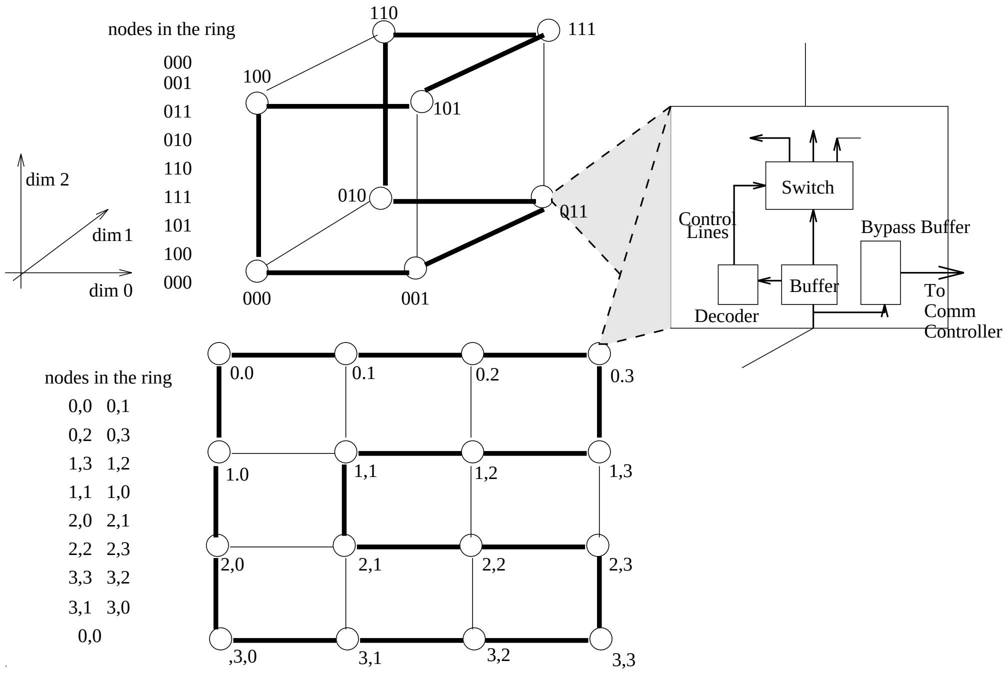


Figure 1: System Configuration

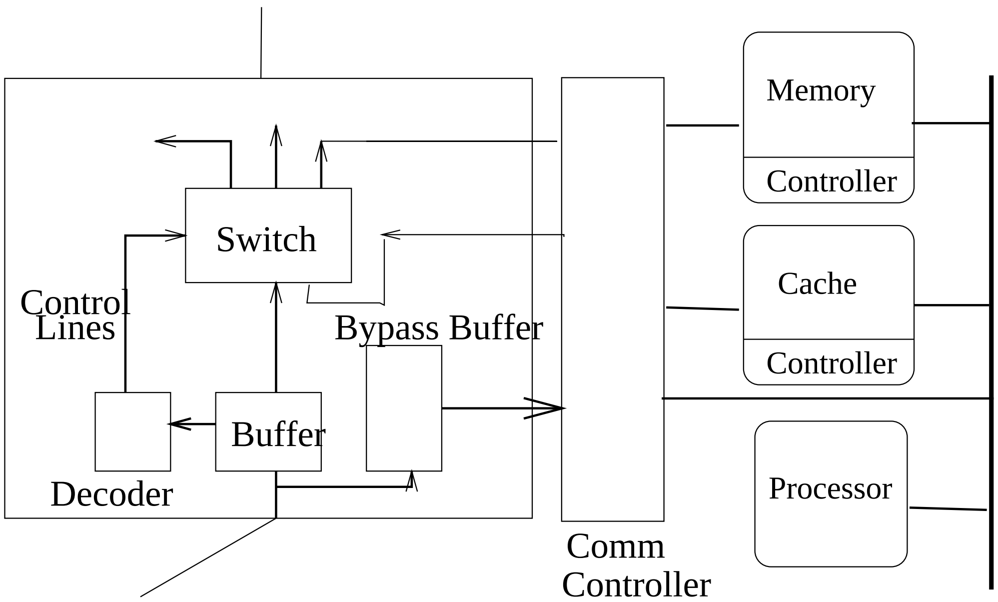


Figure 2: Node Configuration

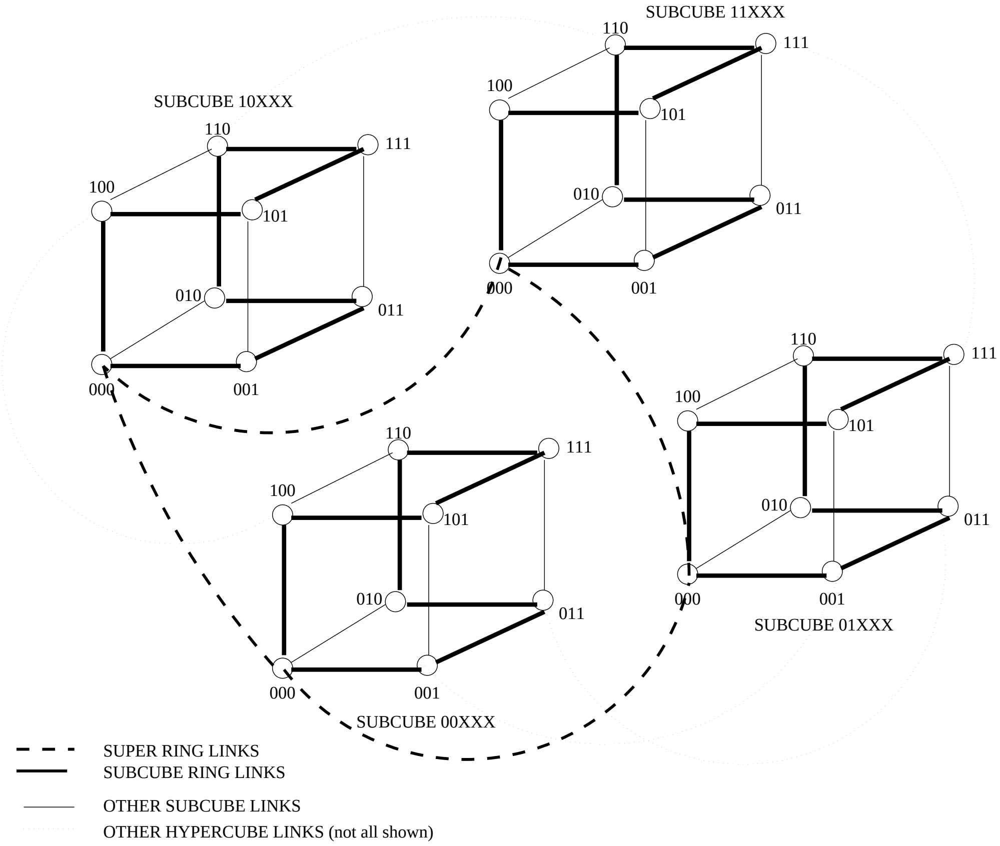


Figure 3: Multiple Ring embedding in a 5-Dim Hypercube

invalidation messages are being sent out from a home node, transmission of half of them is serialized (even though they are pipelined). Also the processing of all the acknowledgments at home is also done one at a time. In such an arrangement the number of links used is also proportional to the number of messages sent, thus increasing the network traffic and delays due to contention. In multicasting using individual messages, the transmission delay can be written to be proportional to the *number of messages sent* $\times$ *length of a message*.

To make the cache coherence protocol efficient, the multicasting of invalidations has to be efficient. One method proposed in this paper is to use a logically embedded ring for sending a single invalidation message. To reduce message communication delays on the ring, wormhole routing is used. In such a case, the ring latency is proportional to *number of nodes in the ring* $\times$ *(one link delay + one node switching delay)*. This ring latency is independent of the number of shared copies present and the number of nodes in the ring is a constant, equal to the number of nodes in the system, for a single ring.

Thus we see that for a small number of invalidations, sending individual messages will incur less delay than sending a message on the ring. For larger number of invalidation messages, the ring would have a lower latency. We have run simulations with varying *average number of read shared copies* and measured the average invalidation time for a write. Figure 4 shows the results of a single ring with zero, two and five pointers compared with a full map scheme. The invalidation time is measured from the time a processor has write miss to the time all copies are invalidated and an invalidation complete acknowledgment is received by the processor from the home node. The full map invalidation time, as expected, increases linearly with the number of read shared copies in the system. Where as all the other three plots show constant time when the number of copies are larger than or equal to the number of pointers kept. We see that for a five dimensional hypercube the crossover point to be 7-8 copies. More detailed analysis is given by Kumar et al in [14].

2.3 Snooping using a Logically Embedded Ring

The proposed scheme introduces a snoopy behavior to a point to point network. As shown in Figure 2, a bypass buffer is added to the input of a link. This is in addition to the one flit buffer already available at each input link. An incoming message is copied into both the one flit buffer as well as the bypass buffer. There is no time penalty for such an arrangement as no extra hardware is included between the input link and the buffer. When the header flit of a message enters the node, it is decoded and depending on the destination, the switch connects the appropriate output link to the input link. If the output link is free the flit is forwarded and the trailing flits of the message follow. In case the destination is the node itself, the message is forwarded to the communication controller of the node. Once a complete message is copied into

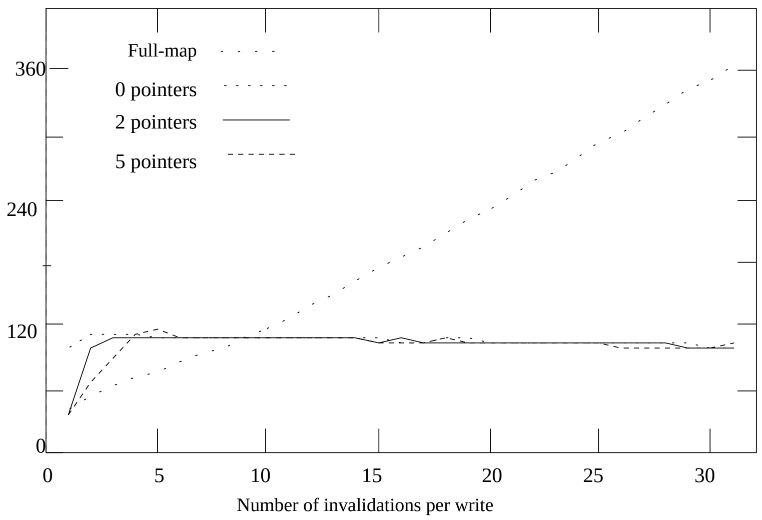


Figure 4: Invalidation Times for a 5 dimensional hypercube

the bypass buffer, a decoder decodes the message type and forwards it to the cache controller or the memory controller through the communication controller. In the case of a broadcast invalidation message traveling a ring, the message is forwarded to the cache controller and the copy in the cache is located and invalidated. When this broadcast message comes back to home, it acts as the acknowledgment itself. This is acceptable because when the ring invalidation message comes back it is guaranteed that all the caches with a copy have received the invalidation message. Thus explicit acknowledgment messages need not be sent to home.

The bypass buffer is required only for the links on the ring, and when a new message arrives one message space in the buffer is allocated and the message is copied into this space. Once the whole message is received and forwarded to the communication controller the space is freed. Also, if it is detected that the message is not a ring broadcast message there is no need to store it in the bypass buffer any further, and the bypass buffer space is freed immediately.

By using multiple rings our method is made scalable. Broadcast delays are reduced by blocking the invalidation message from traversing all the nodes. The nodes in the super ring act as filters, and allow an invalidation message to traverse the sub ring connected to them only if there is a copy of the block available in one of the caches below. This is similar to the coherence protocol proposed for an architecture based on hierarchical buses [15].

There are other methods of broadcasting or multicasting a message. One of them is to use a broadcast tree, the returning acknowledgments are gathered, that is a node waits till all its children send an acknowledgment before it forwards an acknowledgment to its parent node. This method of gather uses packet switching instead of wormhole and also each node in the tree has to process its children's messages thus increasing the processing load as well as the delay. Also proposed in the literature are different multicasting techniques, in these the list of destination nodes is included in the header of the message. These multicasting methods increase the invalidation message length proportionate to the number of destinations. Also the message set up has to be done using a complex hardware circuitry or by the software intervention both of which will increase the message set up and switching time considerably.

3 The Proposed Limited Directory Protocol

As mentioned earlier, the proposed cache coherence scheme is a limited directory scheme. A directory at the memory is kept with number of pointers limited to much less than the number of processors. 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.

- 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.
- The **super nodes** also keep a directory. There is an entry in this directory for each of the cache lines in the sub ring below. If the state of a cache is read-shared or write exclusive in any of the caches below, then the entry in the super node directory is marked present below (PB), else it is marked not present below (NPB). If there is no match for an address in any of the caches below, then there will not be any entry present in the super node directory for that block either.

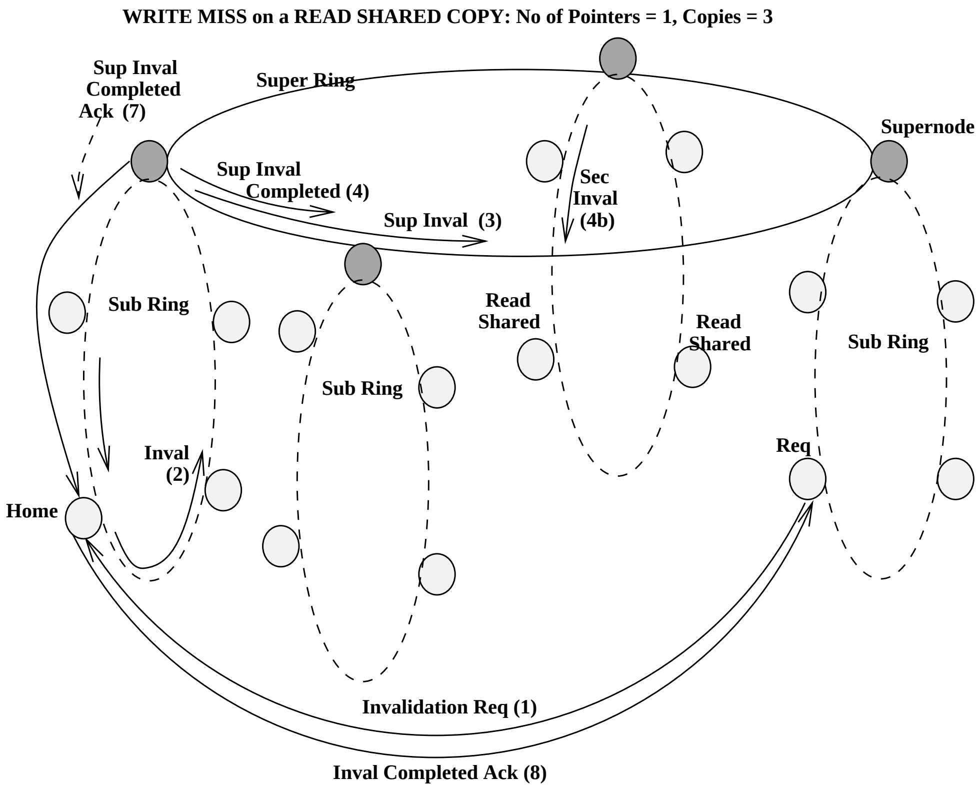
The response of home node to a request depends on the type of request as well as the state of the requested block in the memory.

- 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. In multiple ring case a super node directory update is sent to the super node of the requester. The super node makes an entry in its directory for the requested address block and marks it present below. The super node also sends a super node directory update acknowledgment to home.

- On a write request to a block which is not-cached or Valid, a copy is sent to the requester. The super node directory is updated as in the previous case. In the case of Valid state, an invalidation message is sent to all the caches with a copy. For an invalidation request also, all 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 broadcast bit is set, a single 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. The action in multiple ring case is illustrated in Figure 5. The invalidation message on the sub ring is picked up by the super node and transmitted in a wormhole fashion on the super ring (i.e. second level ring). All the super nodes in this path make a copy of the message and check their super node directory. If a copy is present below then the invalidation message is forwarded over the sub ring connected to this super node. The first super node in addition to forwarding the invalidation message also initiates a super invalidation acknowledgment message. A super node receiving this message forwards it to the next node in the super ring path, only if it has received its sub ring invalidation message back or it did not have to send one at all. When this acknowledgment circles back to the original sender, the super node of the home node, sends a super invalidation complete acknowledgment message to home.

An invalidation process is completed only when home receives its ring invalidation message back and also receives the secondary invalidation complete message from the super node. When a ring message is sent, a separate super node directory update messages need not be sent. The message on the super ring automatically updates the super node directories. For example if an invalidation is traveling the super ring, all the super nodes mark their entry for the cache block as not present below, except unless the requester belongs to its sub ring, in which case an entry is made into the super node directory and marked present below.

- If a cache block replaces a write-exclusive copy from the cache, the copy in the memory has to be updated. For this an update message along with a cache copy is sent to home. For a read copy, a replacement update message can be sent to home, but in case of limited directory schemes if the directory pointer has overflown, this replacement update message may not be very useful. So in our protocol we do not send any replacement updates if the replaced copy is only read-shared. We have also simulated sending of replacements updates, and have



1. Send Invalidation Request from Requester node to Home

2. Home sends a Invalidation message on the ring

3. Home's supernode forwards the Invalidation on the super ring

4. Home's supernode generates a Sup Invalidation Completed message on the super ring

4a. Requesting node's supernode updates its directory

4b. Supernode with a copy Present Below forwards the Invalidation on the sub ring

5. Supernode (S) with a copy Present Below receives its subring Invalidation message back

6. Supernode S forwards the Sup Invaliation Completed message

7. Home's Supernode sends a Sup Invalidation completed Ack to home

8. Home sends an Invalidation Complete ack to Requester

Figure 5: An example of a write miss in multiple ring Scheme

found that in majority of cases not sending replacement updates for read-shared copies performs better.

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 14 in appendix A. The labels on the arc are the messages being sent from one node to another.

If the directory does not keep any pointers, additional messages have to be sent on the ring, these are discussed in appendix B. 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 if 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 as mentioned earlier, 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.

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 [11], has been extended to include our network and cache coherence models. The application programs used are MP3D and Water 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 3000-8000 molecules and ran the simulation for 10 time steps to keep the simulation time manageable. Water, simulates interaction and flow of water molecules. The movement of the molecules is calculated from their potential energy and kinetic energy due to molecular interaction. We used 64-216 molecules with a time step of 0.1×10^{-15} seconds. 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 [16]. 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,sharing is distributed over the whole range in Floyd-Warshall.

In MP3D and Water, too, most of the writes are to the blocks that are shared among a few processors. But in MP3D the number of widely shared blocks is higher and also percentage of sharing is much higher in MP3D than in Water [12]. Matrix multiplication has no sharing for shared writable blocks. In our 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 1. Main memory access time is 4 times slower than cache access time. The channel width is equal to flit length, which means a flit can traverse a link in one processor cycle. The flit length is 2 bytes and the channel width is 16 lines. In a hypercube as the size increases the number of channels increases logarithmically, so in order to keep input/output pins manageable 16 line channel was selected. We also vary it to show the effects of channel width on different protocols. 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 1: 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 (communication controller) in one cycle. Similarly, only one message can be processed by the communication controller in a cycle.

The two types of packets in the system are, the control packets and the data packets. 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 [14]. The data packet also includes a memory block thus its length is assumed to be (8 + Line Size) bytes.

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. We represent our schemes by Dir_iR_n. Where i denotes the number of pointers in the directory and n shows the number of rings in the lower level. R denotes the usage of rings for the broadcast messages. For the full map and limited directory scheme we will use the commonly used Dir_iB notation.

We simulated our Dir_iR_n scheme for various number of pointers (zero to full map) and different number of sub rings (one to sixteen). The full-map scheme is simulated by having as many number of pointers as the number of processors. Dir_iB scheme [4] 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.

Execution driven simulations were run for the five applications described earlier, for 2-6 dimensional hypercubes. The schemes compared in these simulations are full map, Dir_0R_1, Dir_1R_1, Dir_2R_1 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.

5.1 Single Ring Results

The results for MP3D are shown in Figure 6. In this application, for a small system size of 4 or 8 processors Dir_1R_1, Dir_2R_1 behaves similar to full-map. As the system size is increased to 16 and 32 processors, Dir_2R_1 performs better than the full-map, whereas Dir_1R_1 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 the crossover point of Figure 4, the writes for Dir_2R_1 scheme incur less delays than full-map.

On the other hand, Dir_0R_1 performs worse than the full-map scheme. This is because even on a read and write miss on a dirty copy home node needs to send a message on the ring searching for the cache with the write-exclusive copy. This increases the number of messages sent over the ring thus saturating the ring and also increasing the average miss latency. We have also compared the performance of the the

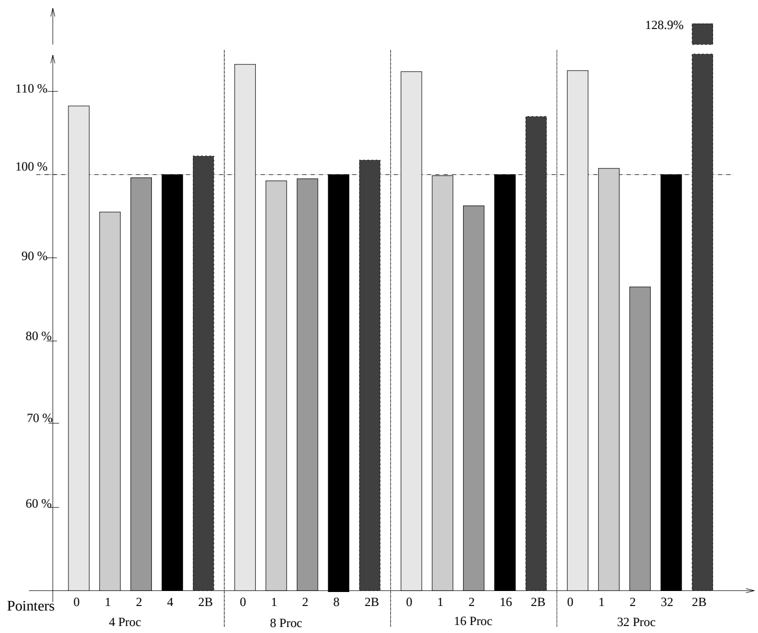


Figure 6: Simulation results for MP3D

traditional implementation of limited directory Dir_2B and show that its performance is worse than the full map, Dir_1R_1 and Dir_2R_1 schemes. The performance of Dir_2B scheme deteriorates with increasing number of processors because of the large delays incurred in multicasting and broadcasting on larger networks.

In FFT, for 4 and 8 processor systems, all the schemes except Dir_0R_1 perform almost same, as shown in Figure 7. Dir_0R_1 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_1 also shows a worsening trend, but Dir_2R_1, 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 at most one invalidation to some other node. In such a case, Dir_2R_1 and Dir_2B and full map would need to send same number of invalidation messages. The behavior of Dir_1R_1 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_1 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_1 execution time decreases compared to the full-map scheme, as shown in 8. Again, the performance of Dir_0R_1 worsens with increase in the size of the system but the deterioration is not as bad as for previous two applications. This is due to the sharing being

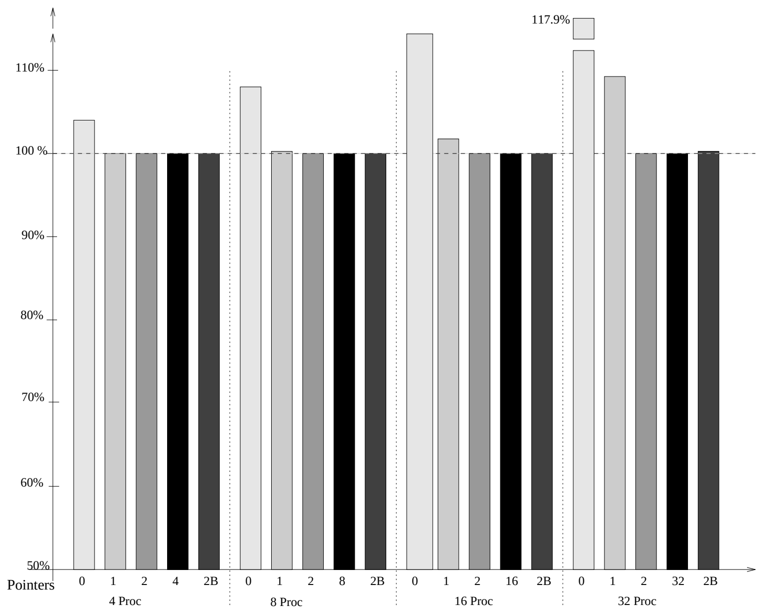


Figure 7: Simulation results for FFT.

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 there is no write sharing at all, except for a small amount 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 9. All the schemes perform equally well in this case. Water results (not shown), also the difference in different methods is not much, still the two and four pointer ring execution time is less than full-map for four and five dimensional hypercubes. This is because the fraction of memory which is shared is small (7% of the writes are to shared blocks) [12].

5.2 Multiple Ring Results

The applications used for multiple ring simulations were MP3D, Water and Floyd-Warshall. The simulations were run for various configurations but results of a 6-dimensional hypercube are shown. The hypercube was divided into 2, 4, 8, and 16 sub rings. And for each case number of pointers was varied from 0 to 4.

Figure 10, shows the results of MP3D on multiple rings. Again as before, the execution time is plotted as percentage of full map. The problem size was scaled up to 8000 molecules. We see for all configurations a directory with 2-4 pointers gives the

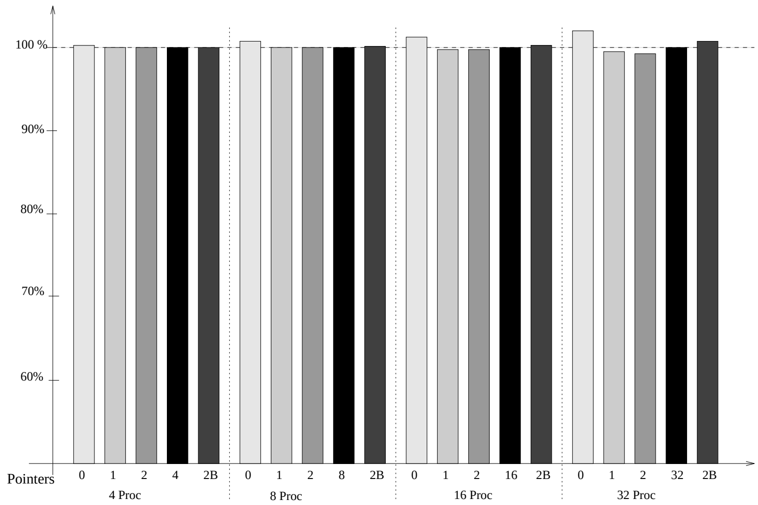


Figure 8: Simulation results for Floyd-Warshall's All Pair Shortest Path Algorithm

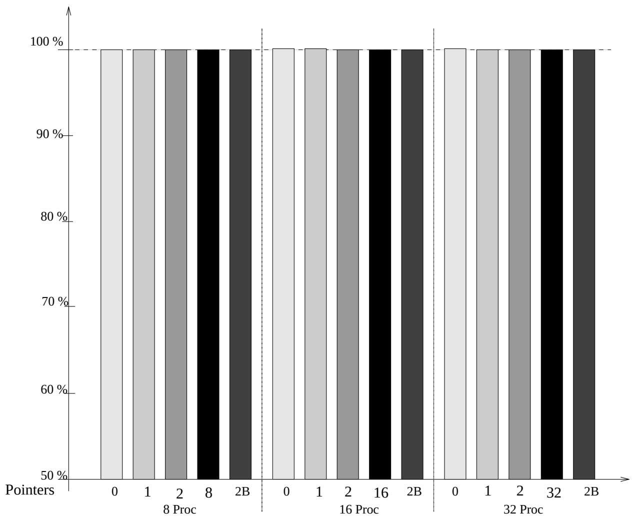


Figure 9: Simulation results for matrix multiplication.

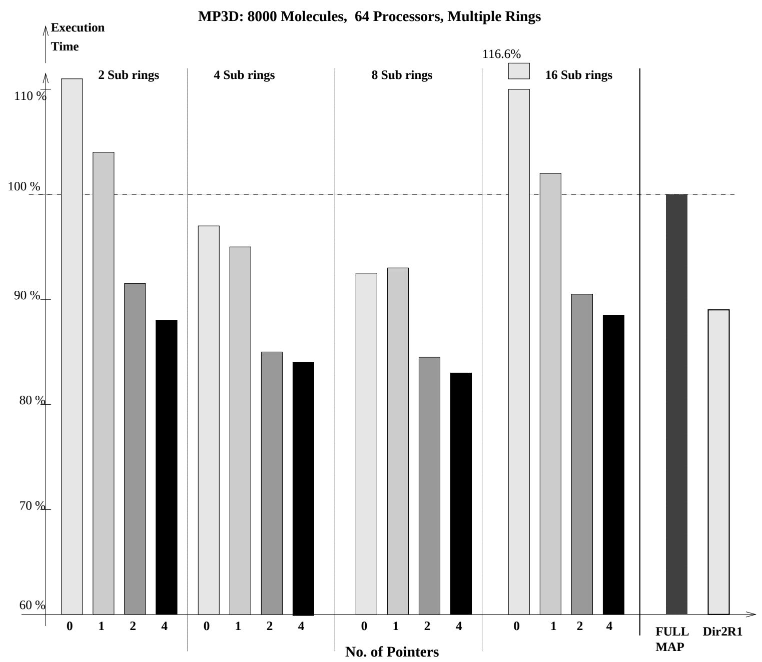


Figure 10: Simulation results for MP3D on Multiple Rings

best results. Also for a 64 processor system, 8 sub rings performance is the best. The 4 sub ring configuration results are also very similar to the eight sub ring ones. But as expected 16 sub rings result in a greater overhead due to larger super ring latency (16 nodes in the super ring) and performance starts to deteriorate. The last bar in the figure is for a single ring with 2 pointers and we observe that eight sub rings with two pointers still performs better than the single ring with two pointers scheme (i.e. $Dir_{2,1}R$). As the size of the system increased from 32 processors to 64, the execution time for two pointer directory method on a single ring goes up from 85% to 89%. We expect the single ring to perform even worse for larger size systems.

The second results presented are for Floyd-Warshall. Due to simulation time and memory constraints the problem was not scaled up as we moved up to 64 processors. We still see in Figure 11 that by using multiple rings we are able to improve on execution time slightly. The times for 2 and 4 pointers on a four and eight sub rings are very comparable to Dir_2R_1 times. We definitely expect the improvement to be greater with scaling up of the problem size with system size.

The third application illustrated here in Figure 12, is Water with 216 molecules. We notice the best configuration to be eight sub rings with two or four pointers. As the number of pointers decreases, even invalidation requests on lightly shared blocks use the rings which increases the average memory access latency. As the number of sub rings is decreased the performance deteriorate, as the size of each sub cube becomes larger thus increasing the ring latencies. Similar to previous results, when the number of sub rings is increased to 16, the ring latency in the super ring becomes too high and

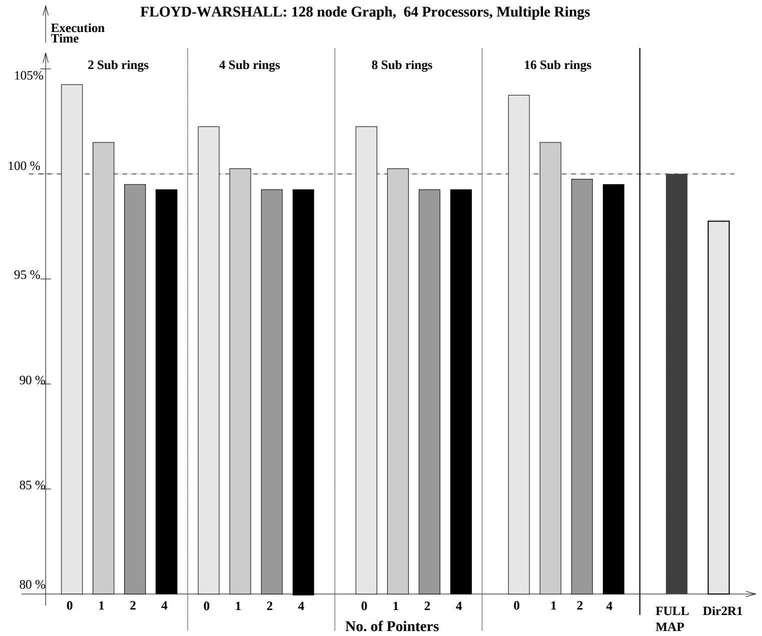


Figure 11: Simulation results for Floyd-Warshall on Multiple Rings

thus results in a higher execution time.

5.3 Effects of Channel Width

The communication delays in a network also depend on number of cycles required to transmit a flit over a link. This further depends on the ratio of the flit size to channel width. Our flit size is 16 bits and a control message length is 64 bits. The results presented so far use a channel width of 16 bits. As discussed in section 2.2 the multicasting using individual messages is more sensitive to the length of the message. For this reason we also experimented with different channel widths to see its effects on the proposed method.

The channel widths used are 8 bits, 16 bits and 64 bits. With increased hypercube size the channel width has to be reduced in order to keep the input/output pins on a switch manageable. 64 bit channel widths are not practical for large system sizes. The results in Figure 13 show that as the channel width reduces to 8 bits from 16, the multiple ring performance deteriorates slightly. This is due to a message taking slightly longer to traverse the ring, thus increasing the ring utilization and contention on the ring. But as the channel width is increased to 64 bits, the multiple ring scheme starts to perform worse than the full map, because of link traversal for an invalidation message is now only one cycle. In this situation the performance of multicasting using individual messages improves much more, where as the ring latency (using wormhole) remains almost same as before. As the system size increases and the channel widths become smaller the proposed multiple ring architecture will still perform very well

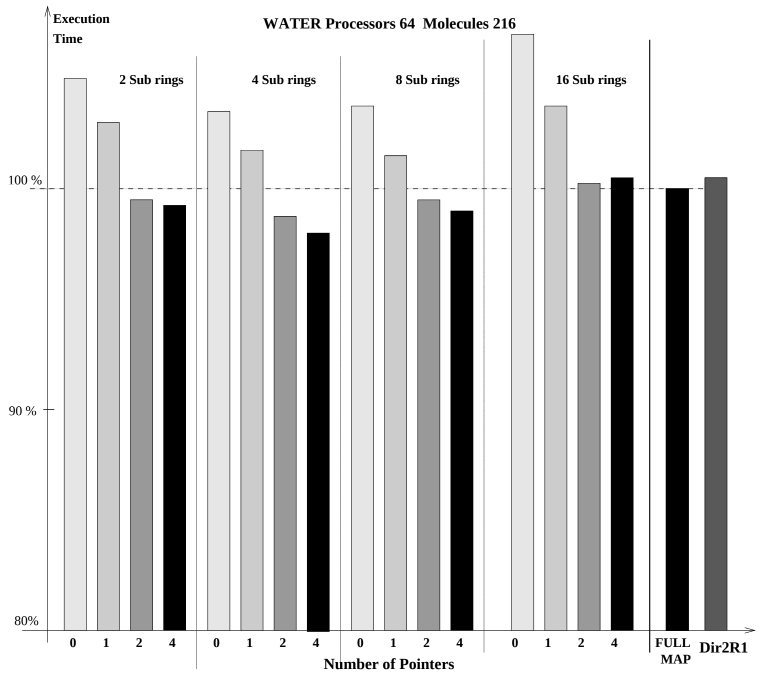


Figure 12: Simulation results for Water on Multiple Rings

compared to full map.

6 Conclusion

In this paper we have proposed an efficient implementation of 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 logically embedded rings. These rings use wormhole routing and provide a snooping capability to a limited directory scheme implemented in a point to point network. The proposed scheme can be realized on any direct network as long as a ring can be embedded on to it. One major problem with shared memory multiprocessor using cache coherence was their scalability to large number of processors. Our multiple ring method with hierarchical organization, is shown to be scalable and also memory access latencies are decreased even though we are using limited directory. The communication over the rings can be seen to be similar to register insertion ring compared to slotted rings used in some earlier proposed methods [8, 9, 10]. Most of the traffic in cache coherence environment is bursty, instead of uniformly distributed over time. Under such load conditions (i.e. low average load) an insertion ring would definitely perform better than slotted ring [17].

Exhaustive simulations have shown that for different types of sharing patterns, from

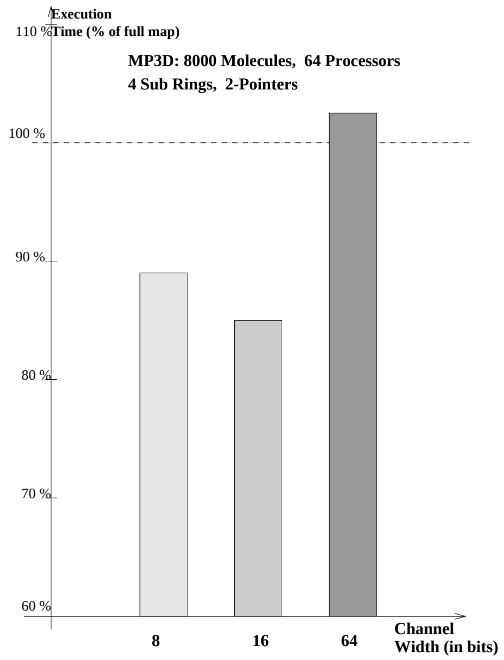


Figure 13: Simulation results for MP3D for different channel width

no sharing in matrix multiplication to widely distributed sharing in Floyd-Warshall and sharing at the low and high end in MP3D, 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. Though it may seem counter intuitive, but the results shown in Figures 6–9 suggest that the performance of the proposed ring based scheme improves against full-map as the system size increases. This can be explained by the faster invalidation scheme used by the ring based scheme compared to individual message based invalidation used by full-map, as discussed in Section 2.

One could argue that the full-map scheme could use an efficient broadcasting and multicasting scheme to achieve faster invalidation. Such schemes are either packet switched based requiring processing at intermediate nodes or need complex hardware support with increased latencies. Our next step is to evaluate the proposed scheme against such broadcast and multicast schemes [18, 19].

References

[1] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," *Proc. 10th Ann. Int'l. Symp. Comp. Arch.*, pp. 124–132, 1983.

[2] Q. Yang, L. N. Bhuyan, and B. C. Liu, "Analysis and comparison of cache coherence protocols for a packet switched multiprocessor,," *IEEE Trans. Comput.*, vol. C-38, no. 8, pp. 1145–1153, Aug. 1989.

[3] 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.

[4] 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.

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

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

[7] 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.

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

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

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

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

[12] J. P. Singh, W. D. Weber, and A. Gupta, "Splash: Stanford parallel applications for shared memory," *ACM SIGARCH Computer Architecture News*, vol. 20, , March 1992.

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

[14] 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.

[15] Q. Yang, G. Thangadurai, and L. N. Bhuyan, "Design of a adaptive cache coherence protocol for large scale multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, pp. 281–293, May 1992.

[16] P. K. Mannava, A. Kumar, and L. N. Bhuyan, "A new limited directory cache coherence scheme for shared memory multiprocessors," Technical Report TR-95-014, Dept of Comp Sc, Texas A & M Univ., 1995.

[17] L. N. Bhuyan, D. Ghosal, and Q. Yang, "Approximate analysis of single and multiple ring networks," *IEEE Transactions on Computers*, vol. 38, no. 7, pp. 1027–1040, July 1989.

[18] P. McKinley, H. Xu, A. Esfahanian, and L. Ni, "Unicast-based multicast communication in wormhole routed networks," In *Proceedings ofInternational Conference on Parallel Processing*, 1992.

[19] D. K. Panda, S. Singal, and P. Prabharkaran, "Multidestination message passing mechanism conforming to base wormhole routing scheme," *(PCRCW-94) Lecture Notes in Computer Science*, no. 853, pp. 131–145, 1994.

Appendix A: Description of messages

Message Type	Request Type	Condition
Cache to Home	Read/Write/Invalidation requests Ack of receipt of requested data Acking inval of a read-shared copy Replacements	On a cache miss
Home to Cache	Requested memory block Ack for completion of invalidations	
Cache to Cache	Forwarding cache block	From write-ex copy to req (Dir_0R_n case)
Ring Messages	Invalidation message Search/SearchI for a write-ex copy	No of Shared copies $> i$ for Dir_0R_1 case
Super Ring* Messages	Invalidation and ack messages SSearch/SSearchI	No of Shared copies $> i$ for Dir_0R_n,and $n > 1$
Secondary Ring* Messages	Similar to Super Ring Messages	copy present in the lower level ring for Dir_0R_n
Super and Home*	supernode dir update acks for invalidation and updates	for Dir_0R_n $n > 1$

TABLE 2: Message Categorization

* Used only in Multiple Rings case

The messages sent in the single ring case are categorized as cache to home node, home node to cache, one cache to another, and ring messages (as shown in Table 2). The first three of these use the usual shortest path routing from source to destination. The ring messages as the name implies use the embedded ring. In multiple rings, in

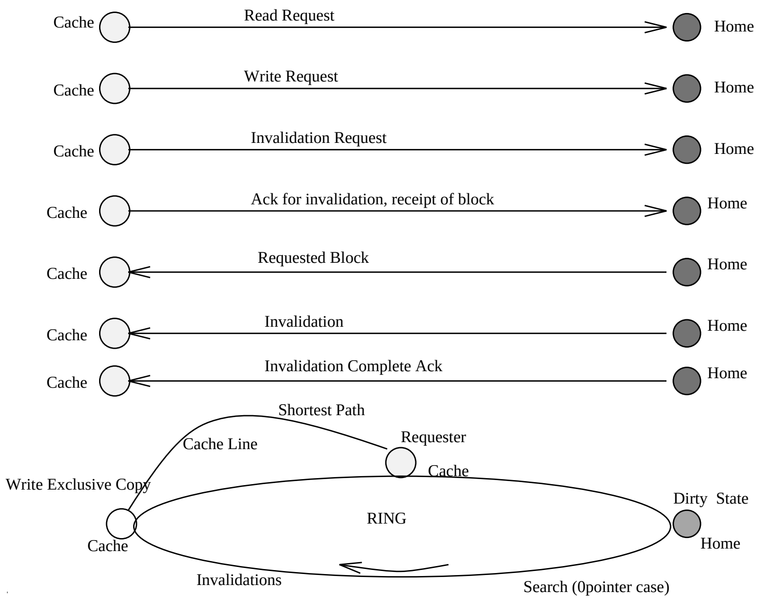


Figure 14: Different Message Types in a Single Ring Scheme

addition to the above, super ring, secondary ring and super node to home and home to super node messages are also used. 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. A home node sends a message to super node to update its super node directory. This is generated in response to a cache below the super node directory generating a memory request. After the super node updates its directory it acknowledges this by sending a ack message to home.

Appendix B: Additional Messages for Zero Pointer Case

Additional ring messages need to be used for a directory with zero pointers. On a read request to a dirty copy, a search message is 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 of the block 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. In multiple ring case, the search message is forwarded on the super ring, and each super node checks its super node directory and if the block is present below the search message is forwarded onto the sub ring. As the block is in a dirty state only one cache has a write exclusive copy and the search has to proceed at most on one sub ring. When this sub ring search comes back to the

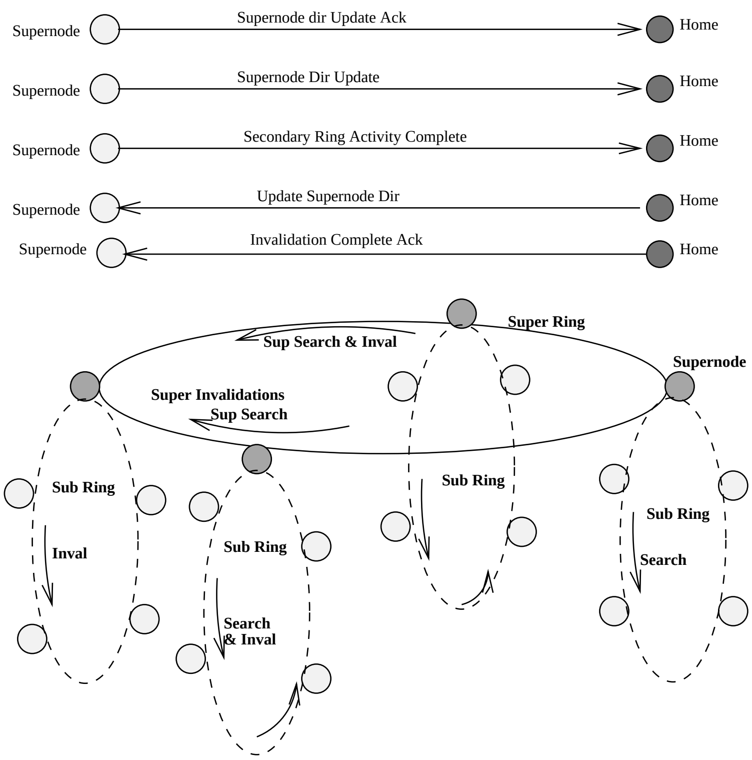
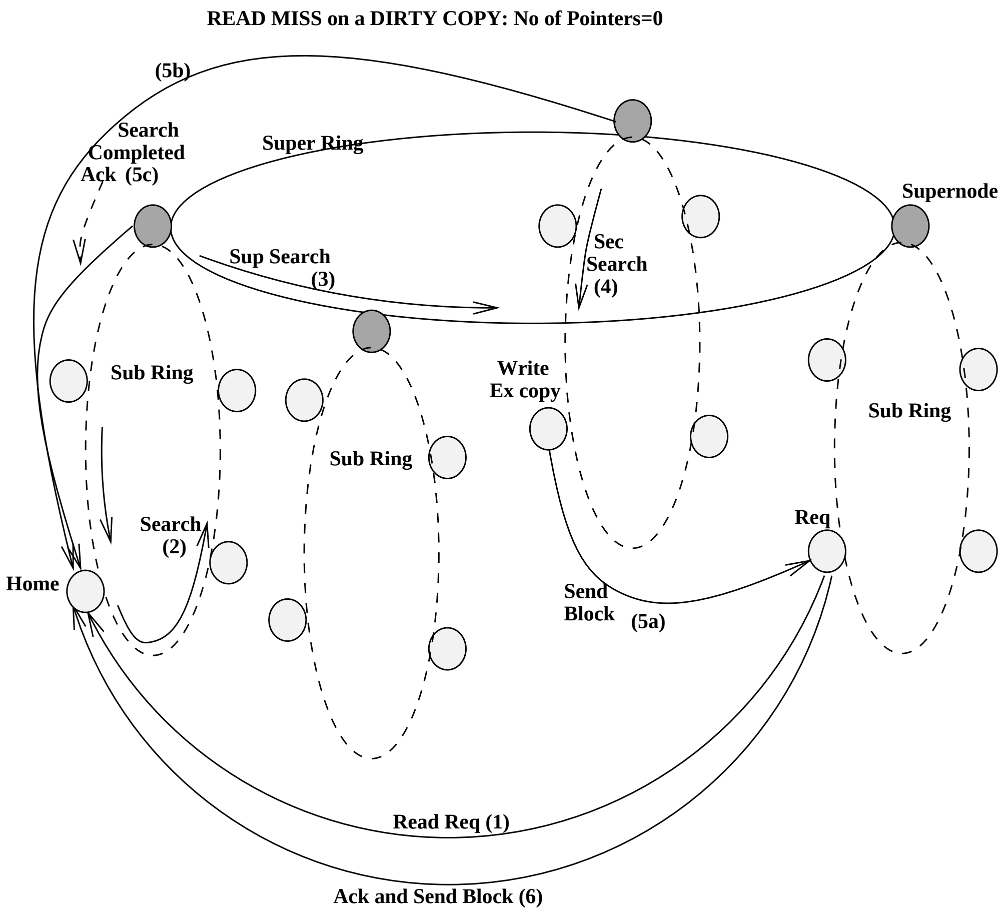


Figure 15: Different Message Types in the Multiple Ring Scheme

super node, an search complete acknowledgment is sent to home. The memory access activity is not completed till home receives a copy of the block from the requester and also receives the sub ring and super ring search complete acknowledgments. Figure 16 illustrates a read request on a Dirty copy with zero pointers on a multiple ring configuration. A similar procedure is followed when a write request is received for a dirty copy.



1. Send Read Request from Requester node to Home

2. Home sends a Search message on the ring

3. Home's supernode forwards the search on the super ring

3a. Requesting node's supernode updates its directory

4. Write Ex node's supernode forwards the search on to the subring

5a. Write Ex node forwards a copy of the block to the requester

5b. Write Ex node's supernode sends search complete ack to home

5c. Home's supernode also sends search complete ack to home

6. Requester sends an acknowledgement to home

Figure 16: An example of a read miss in multiple ring Scheme