Published in ICPP'91

CACHE COHERENCE ON A SLOTTED RING

Luiz A. Barroso and Michel Dubois
EE-Systems Department
University of Southern California
Los Angeles, CA 90089-1115

Abstract -- The Express Ring is a new architecture under investigation at the University of Southern California. Its main goal is to demonstrate that a slotted unidirectional ring with very fast point-to-point interconnections can be at least ten times faster than a shared bus, using the same technology, and may be the topology of choice for future shared-memory multiprocessors. In this paper we introduce the Express Ring architecture and present a snooping cache coherence protocol for this machine. This protocol shows how consistency of shared memory accesses can be efficiently maintained in a ring-connected multiprocessor. We analyze the proposed protocol and compare it to other more usual alternatives for point-to-point connected machines, such as the SCI cache coherence protocol and directory based protocols.

1. Introduction

It is a well known fact that bus-based architectures, which are the most popular topologies in current commercial systems, have serious electrical problems that have kept them from reaching higher bandwidths, thus limiting their scalability [4]. Moreover, processors clocked at 40 MHz are already available on the market, and we believe that even a small number of these could generate enough traffic to saturate current interboard buses, in a caching shared-memory multiprocessor. On the other hand, point-to-point unidirectional interconnections can be made much faster than bus interconnections since they do not require arbitration cycles, and lack most of the electrical problems with shared buses, namely signal propagation delays and signal reflection/attenuation. Current IEEE standard proposals are based on 500 Mbps, 16 bits wide unidirectional point-to-point interconnects [17]. The *Express Ring* is a cache-coherent shared-memory multiprocessor with the simplest type of point-to-point interconnection network: a unidirectional ring. We believe that such a ring can be made at least ten times faster than state-of-the-art buses, therefore showing much better scalability.

The use of local caches in a shared-memory multiprocessor greatly enhances overall memory access performance, since a large fraction of processor references is likely to be present in the cache, and is resolved at processor speed. Private caches also affect performance by decreasing the traffic in the interconnect, thus avoiding conflicts and queueing delays. However, in order to enforce a consistent view of multiple cached copies of shared writable data, one requires a c*ache coherence protocol*. In this paper we develop a cache coherence protocol for a slotted unidirectional ring multiprocessor. We believe that the proposed protocol is a simple and efficient solution for the coherence problem in a ring architecture, and we compare it with other protocol options, such as directory based schemes [18][3] and the SCI cache coherence protocol [12].

This work is funded by NSF under Grant No. MIP-8904172. Luiz A. Barroso is also supported by CAPES/MEC, Brazil, under Grant No. 1547/89-2.

Snooping protocols [8][13] are very popular in bus-based systems, mainly because of their simplicity and low cost of implementation. Unfortunately this class of protocols relies heavily on the broadcasting of information, and is inadequate for most point-to-point interconnected systems, where broadcasting is generally very expensive. Several cache coherence protocols for point-to-point interconnected machines have recently been proposed in the literature [18], most of them making use of directories to keep track of the copies for every memory block. These schemes are relatively complex to implement. Because of the natural broadcasting structure of the Express Ring, we believe that an adapted version of a snooping cache coherence protocol performs better than a directory based mechanism, with the advantages of being simpler to implement and scale to larger configurations.

In the next Section we briefly describe the Express Ring architecture, its pipeline features and memory organization. In Section 3 we present a cache coherence protocol for this architecture, including some implementation issues. The performance characteristics of the proposed protocol are presented in Section 4, in which we also compare it with other alternatives in the context of the Express Ring architecture. Final conclusions are drawn in Section 5.

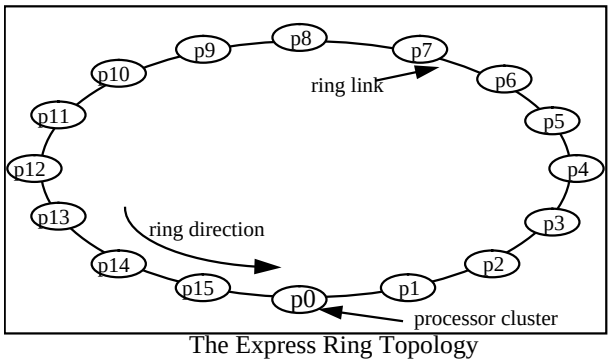
2. The Express Ring Architecture

The Express Ring intends to demonstrate that a high-speed slotted unidirectional ring is not only feasible but may be superior to a shared bus as the interconnection structure for shared-memory multiprocessors. Our design philosophy was to keep the interconnection structure very simple, in order to be able to clock it as fast as current technology allows. We also wanted to avoid any centralized arbitration, as well as a complex interconnection access control

mechanism.

An important feature of the Express Ring architecture is that, as far as the programmer is concerned, the system behavior is identical to a bus-based architecture, since all ring transactions experience the same latency. This characteristic, together with the use of relatively large private caches, makes the Express Ring suitable for general purpose computing.

The Express Ring architecture, which is schematically shown in Figure 1, consists of a set of high-performance processor clusters interconnected by a slotted ring structure. Each processor cluster, as depicted in Figure 2, consists of 2 processors connected to a shared dual-port cache, a fraction of the system's physical memory space (including main memory and swapping disk space) and the ring interface. Transfers on the ring are synchronized by the same clock, but they are asynchronous to the processors operation.

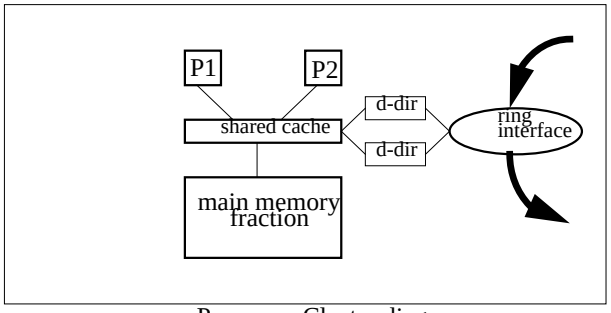


The Express Ring Topology

Figure 1.

Using a ring topology to interconnect processing elements is not a new idea. Extensive research have been done in the design and analysis of ring-type interconnections for local area networks (LANs), such as the token ring [11] and the Cambridge slotted ring [10][16] network. Ring interconnections have also being studied in the context of more tightly-coupled systems, as in the CDC CYBERPLUS system [7] and the MIT Concert Multiprocessor [9]. However, we are not aware of the use of a slotted ring network as the interconnection of a shared-memory cache coherent multiprocessor system. The main benefit of the slotted ring is that several messages can be transmitted at the same time, which increases the communication bandwidth.

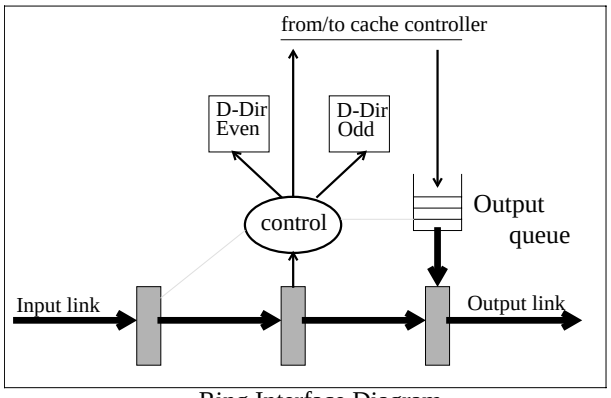
The Express Ring interconnection network can be viewed as a circular pipeline, with 3*P stages, where P is the number of processor clusters (or nodes) in the system, i.e., each processor cluster accounts for 3 pipeline stages, as shown in Figure 3. At each ring clock cycle, 64-bit wide packet slots are transferred from a stage of the pipeline to the next one. Messages are inserted in the ring by filing packet slots with useful information, and removed by marking a packet slot as empty. There are 3*P packet slots circulating in the ring. The pipeline clock can be extremely fast, since no



Processor Cluster diagram

Figure 2.

centralized arbitration is required for the packet slots: a processor cluster may use the first empty slot that passes through it to send a message. Such a scheme may lead to starvation of a cluster, specially when the ring traffic is relatively heavy, and the utilization of the slots approaches 100%. A simple and effective mechanism to prevent starvation, without requiring any central arbiter or reservation of slots, is discussed in Section 3.



Ring Interface Diagram

Figure 3.

Routing in this architecture is very straightforward. An incoming message is either consumed by the cluster or forwarded to the next cluster in the ring order. The ring interface has a very fast logic that scans the first bits of each message and determines whether it has to be forwarded or removed from the ring. A block diagram of the ring interface is shown in Figure 3. It is important to note that the speed of the forwarding logic is critical in the Express Ring design, and is likely to become the bottleneck on the pipeline clock frequency.

There is a single physical address space shared by all processors in the system. The memory is distributed among the clusters in the ring and the location of a memory block is determined by the higher address bits. The processor cluster to which a block is mapped is called the block's *home cluster*. Swapping space is provided by disks attached to some of the clusters. A single *dirty* bit, is associated with each cache block in main memory. The dirty bit is used to indicate wether the current main memory copy of a block is up-to-date, and it is critical to the performance of the

proposed coherence protocol.

A dual-directory serves coherence requests coming from the ring; it contains a copy of the local cache directory (tags + state information), so that snooping activity does not interfere with normal processor accesses to the cache. The dual directory is 2-way (even/odd) interleaved to be able to keep up with the rate at which coherence messages arrive from the ring. Caches may contain blocks that map to the local cluster's physical space or to any other cluster's memory. Both cache misses and invalidations may generate messages to the ring in order to satisfy local accesses and maintain global system coherence.

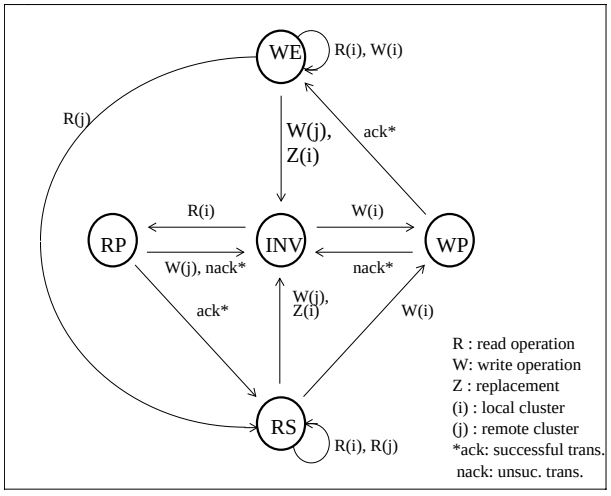
3. The Express Ring Cache Coherence Protocol

3.1 Description

In this Section we present a snooping cache coherence protocol for the Express Ring architecture. We propose a write-back/write-invalidate protocol with five cache block states and two memory block states. Three of the cache block states, *Invalid* (INV), *Read-Shared* (RS) and *Write-Exclusive* (WE), are taken from the basic write-back/ write-invalidate snooping protocol. A cache block in INV state is assumed to be not present in the cache. A cache block in RS state can be accessed for reads but not for writes; several RS cached copies of a block may exist at the same time. A cache block in WE state can be accessed for both reads and writes, and the protocol ensures that when a block is in WE no other cache has a valid (WE or RS) copy of it.

Transitions among these three states in a bus-based system are virtually conflict free, since the bus arbitration mechanism itself takes care of imposing an order among global coherence events. However, the situation in the Express Ring is not so simple. In this system several messages can be traversing the ring at any given time, and two or more clusters may be attempting to access the same block concurrently, causing a conflict. Conflicts may have to be solved by forcing one or more clusters to abort their state transitions. In our protocol we represent state transitions that are subject to conflicts by two extra cache states: *Read-Pending* (RP) and *Write-Pending* (WP). We identify such transitions with specific cache states because it simplifies the protocol description, but it is important to note that pending states do not have to be coded explicitly in the cache directory. Registers in the snooper could keep track of blocks to which transactions are pending in the ring. In order to support lockup-free caches it is necessary to allow more than one cache block to be in a pending state in a cluster at the same time [5].

A cache block in RP (WP) state is in transition to a RS (WE) state but has not yet committed, i.e. has not yet received the corresponding acknowledgment. An acknowledgment for a miss is a message that provides the missing block, and the acknowledgment for an invalidation is a signal that indicates that all system caches successfully invalidated their copies of a block. A cache block in RP (WP) state may also abort its transition to RS (WE) state if some kind of negative acknowledgment is received. We discuss the acknowledgment mechanisms in detail later in this section. The diagram in Figure 4 shows the possible states of a block in a given cache (i). W(i) and R(i) denote respectively write and read operations generated at the local cluster (i), and (j) denotes any cluster other than (i).



State transition diagram of a block in cache (i)

Figure 4.

The two memory states indicate wether the current main memory copy of a block is valid or not, and they are implemented by a dirty bit, as described in the previous section. The memory copy of a block is not valid when some system cache has a copy of that block in WE state. We note that when a block's memory state is valid, there may or may not be cached copies of it. The two memory states are named *modified*/*unmodified*.

Coherence actions are procedures that have to be enforced by the coherence protocol in order to preserve an overall consistent view of the shared memory. Coherence actions are triggered by one of the following events: read miss, write miss, invalidation and replacement. Below we explain the protocol behavior in further detail by describing the state transitions and data movement involved in every coherence action.

Read Miss: A read miss immediately changes the cache state of the missing block from INV to RP. The requesting cluster's cache state changes to RS when the missing block is received. If the memory block state is *unmodified* the home cluster provides the block to the requesting cluster, and the memory block state remains *unmodified* after the transition is completed. Otherwise the *dirty cluster* (a cluster with the WE copy) sends the valid copy to the requesting cluster and to the home cluster, so that the memory copy is also updated. The cache state in the dirty cluster changes to RS and the

memory state changes to *unmodified*.

Write Miss: The local cache state of the missing block goes immediately to WP, and changes to WE when a valid copy of the block is received. In a write miss, not only a copy of the block has to be fetched but also all other existing copies have to be invalidated. The block is provided by the home cluster if the memory state is *unmodified* or by the dirty cluster, if the memory state is *modified*. The final cache state in all clusters other than the requesting one is INV, and the final memory state is *modified*.

Invalidation: Generated when a processor attempts to modify a block that is present in its cluster's cache in RS state. An invalidation action ensures that the cluster first acquires an unique copy of the block before it is allowed to modify it. The requesting cluster cache state goes immediately to WP, and changes to WE when an acknowledgment is received. One should note that invalidations only happen when the memory state is *unmodified*. The final cache state in all clusters other than the requesting one is INV, and the memory state changes to *modified*.

Replacement: Generated when a cache needs the block frame that is presently occupied by a block in WE state (i.e., a block that has been altered). The block has to be *flushed* to the home node, so that the updates are not lost. The requesting cache state goes immediately to INV, and the memory state changes to *unmodified*. Blocks in any other cache state (WP, RP, RS and INV) do not have to be written back since they have not been altered.

3.2 Implementation

We now discuss some implementation issues of our protocol, also detailing its behavior in conflicting situations. The protocol is implemented as a set of *coherence messages* exchanged through the slotted ring in order to implement the coherence actions described earlier. A miss in a cache block that is mapped to the local cluster's physical space (i.e., the requesting cluster and the home cluster are the same) is called a local miss. A local read miss does not generate a coherence message to the ring if the block's memory state is *unmodified*. In all other situations misses, invalidations and replacements cause coherence messages to be sent to the ring. A summary of the protocol coherence messages is shown in Table 1.

3.2.1 Coherence Messages

Responses to a *Read-Block* message (generated when a read miss occurs) vary depending on the current memory state of the block. If the block's memory state is *unmodified*, the home cluster contains a valid copy and provides it to the requesting cluster by replying with a *Send-Block* message. However if the block's memory state is *modified*, the copy has to be provided by the dirty cluster. In this situation, the dirty cluster replies with a *Send-Block-Update* message that contains the requested block. The requesting cluster forwards a *Send-Block* message to the block's home node when it receives the *Send-Block-Update* message, so that the main memory can be updated.

Table 1.
Protocol Commands/Responses

Event	Command	Response
read miss	Read-Block	Send-Block Send-Block-Update
write miss	Read-Exclusive	Send-Block
invalidation	Invalidate	(none)
replacement	Send-Block	(none)

A *Read-Exclusive* message is generated when a write miss occurs. Either the dirty cluster (if the block is *modified*) or the home cluster (if the block is *unmodified*) replies to a *Read-Exclusive* message by providing a valid copy of the block. In this case, the reply is always a *Send-Block* message, even if the block's memory state is *modified*. In other words, we do not update the main memory copy of a block when a WE copy simply migrates from one cluster to another. A Read-Exclusive message also invalidates every other copy of that block in the ring.

An *Invalidate* message is generated when a cluster attempts to write to a block in RS state in its local cache. In this case there is no need to fetch a copy of the block, but it is important to invalidate all other cached copies of it.

3.2.2 Message Formats

There are two message formats on the ring. *Read-Block*, *Read-Exclusive*, and *Invalidate* are very short messages, named *probe* messages, with the following format:

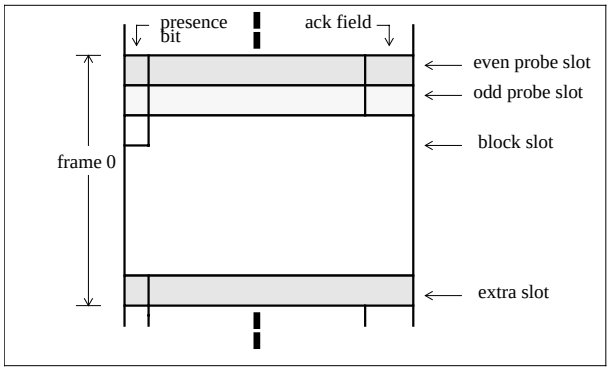
|<message type>|<requester addr>|<block addr>|<ack>| = 44 bits
4 6 32 2

Send-Block and *Send-Block-Update* messages (called *block* messages) are larger since they carry a cache block. Here we consider a cache block size to be eight 32-bit words (256 bits). The format of the block messages is:

|<message type>|<requester addr>|<block addr>|<block>| = 298 bits
4 6 32 256

Probe messages fit in a single 64-bit packet, and occupy only one pipeline stage (a *probe slot*) in the Express Ring, while block messages occupy 5 consecutive pipeline stages (a *block slot*). Probe slots and block slots are arranged in *frames*, that circulate through the slotted ring. Frames are composed of one probe slot for blocks with even addresses (even probe slot), one probe slot for blocks with odd addresses (odd probe slot), one block message slot (which have an even or odd address), and an extra slot used for

intercluster interrupt signals. The frame format is shown in Figure 5.



Frame organization

Figure 5.

We say that a given slot is empty if no valid information is currently being transmitted through it. A cluster wanting to transmit a probe has to wait until it detects an empty probe slot of the same parity. A block message may be transmitted using any empty block slot. There are three basic motivations to arrange probe and block slots in frames:

(1) Enforce a minimum interarrival time for probes with same parity, giving enough time for the interleaved dual-directory to respond. Using frames, the minimum interarrival time for probes with same parity will be eight ring clock periods, which gives 40 ns. (ring clock freq. of 200 MHz) for the dual-directory to respond, instead of 5 ns.

(2) Eliminate the problem of having to deal with two different sizes of messages on the ring, which simplifies arbitration. If any pipeline slot could be used for probe and block messages a cluster waiting to transmit a block message would not be able to determine whether there are enough consecutive empty slots to transmit its message when it sees an empty slot. Transmitting block messages in non-consecutive pipeline slots, or temporarily removing messages from the ring to accommodate a block message, would make the ring interface much more complex, and that would certainly slow down the ring clock.

(3) Provide an efficient acknowledgment mechanism for probes. The need for such mechanism will become clear in Section 3.2.3.

A probe message traverses the whole ring exactly once, being consumed by the cluster that issued it. This is necessary because all the clusters' directories have to be consulted on a miss or an invalidation. However, a block message does not circulate through the whole ring, but only through the path between its source and destination. Assuming balanced access patterns, a block message traverses one half of the ring on the average, and this is the reason why we have chosen to put two probe slots and one block slot per frame. The sender of a probe and the destination of a block message are responsible for removing them from the ring. A message is removed by resetting its presence bit (see Figure 5).

When a probe message passes through a cluster it generates a dual-directory lookup but does not wait for the result of the lookup; rather, it is forwarded to the next cluster without delay. Any acknowledgment for a probe uses the <ack> field of the next probe of same parity. This mechanism is included to resolve conflicting situations that may arise when probes for the same block are issued by distinct clusters in a very short time interval. It also permits a cluster to reject a probe.

3.2.3 Conflict Resolution

Conflicting situations occur when more than one cluster issue a probe for a given block in a very short time interval. The protocol has to be able to resolve these situations not only keeping the global state of the block consistent, but also avoiding deadlocks, livelocks and starvation of a cluster. To be able to deal with such situations there is an <ack> field in every probe slot. Acknowledgments for an even (odd) probe in frame i will be put in the <ack> field of the even (odd) probe of the subsequent frame (frame i+1). The reason why acknowledgments for a probe are not "piggybacked" in the same probe slot is to allow the ring interface logic enough time to respond. The <ack> field consists of two bits. Those bits are defined as:

a-bit : acknowledgment bit. Set by the home cluster or the dirty cluster of the block.

ns-bit: no-snoop bit. Set by a cluster that was not able to consult its dual-directory in time to acknowledge a probe.

Both the a-bit and the ns-bit are reset by the requesting cluster when a probe is sent. The a-bit is set by the cluster that has the valid copy of a block being requested by a *Read-Block* or a *Read-Exclusive* probe (which can be either the home cluster or the dirty cluster), signaling that the request was received and that the requested block will be provided. When the requesting cluster sees the probe returning, it removes it from the ring and waits for the acknowledgment that comes in the next frame. If the a-bit is set it waits for the requested block to arrive (still in a pending state) to complete the transition, otherwise it re-issues the probe in the next available slot. When a *Read-Exclusive* or an *Invalidate* probe for a given block passes through a cluster that has that block in RP state, that block's transition is immediately aborted.

Another hazardous situation occurs when multiple *Invalidate* or *Read-Exclusive* probes are issued concurrently. In this case, the cluster with the valid copy serves as the *arbitrator cluster*. The probe that reaches the arbitrator cluster first has its a-bit set. The cluster that receives the probe with the a-bit set "wins" the arbitration and is allowed to write in the block (i.e., change from WP to WE state). The

remaining clusters receive the a-bit reset and have to abort the request. Note that a cluster that aborts an *Invalidate* request must retry with a *Read-Exclusive* probe, since it no longer can assume that its previous RS copy of the block is valid.

The ns-bit is set by a cluster that has its dual-directory busy, and cannot accept a given probe. This bit may not be needed if the engineering of the ring interface always allows enough time to snoop at any request. The behavior of the protocol when the ns-bit is necessary is well explained in [1].

While we do not think that fairness will be a serious problem in the Express Ring interconnection, it is possible that a given cluster has to wait an arbitrary long period of time until it finds an empty slot to send a message. If that turns out to be a problem the following policy can be adopted. We simply enforce that every time a cluster removes a message from the ring it does not use that slot but passes it to the next cluster, even if it has messages to transmit. This mechanism works like a token passing scheme, where the token is actually the empty slot. The upper bound on the time to acquire an empty slot of any kind will be a small multiple of the total ring traversal time, since there are multiple frames in the slotted ring.

4. Protocol Evaluation

In this section we evaluate the expected latencies to satisfy coherence messages in the Express Ring cache coherence protocol, as well as in a directory based protocol and in the SCI cache coherence protocol. We compare the three protocols in the context of the Express Ring architecture. Brief descriptions of centralized directory and the SCI protocols are given below.

4.1 Centralized Directory Protocols

In this class of protocols [14] every coherence message is directed to the block's home cluster, which keeps track of the state and location of all cached copies of the block. In directory protocols, the home cluster allows sharing of a block for read-only copies but enforces exclusive access for writable copies. The home cluster satisfies all misses if there is no writable (dirty) copy of the block, forwarding the request to the dirty cluster otherwise. When receiving an invalidation, the home cluster sends invalidation messages selectively only to the clusters sharing a particular block, and replies to the requesting cluster once all copies have been invalidated. The main difference between directory and snooping schemes is that directory-based protocols do not rely on the broadcasting of coherence information.

4.2 SCI Cache Coherence Protocol

The SCI standard, which is being defined by IEEE [17], proposes a type of distributed directory protocol in which a linked list of clusters is maintained for each cached block, called the *sharing list*. The list is implemented by forward and backward pointers in each cache block frame. Coherence requests are sent to the home cluster which satisfies them if the block is currently not cached. If the block is being cached, the home cluster forwards the request to the cluster at the head of the sharing list for that block, which takes the appropriate coherence actions and responds to the requester. In case of a read miss, the missing cluster is inserted at the head of the list either by the home cluster or the previous head. In case of a write miss or an invalidation, all clusters are sequentially removed from the sharing list until the writing cluster is the only one left.

4.3 Latency Analysis

In analyzing the latencies we differentiate miss latencies from invalidation latencies. The latency to satisfy write misses, which can be seen as both a miss and an invalidation, can be approximated by whatever action takes longer, e.g., fetch the missing block or invalidate current cached copies. We will further assume that directory and SCI protocols make use of message formats similar to our probes and block messages, so that the latencies being considered here are actually probe message round-trip latencies.

Miss Latency

In our snooping protocol, only one ring traversal is required to satisfy a miss. A probe is inserted and removed from the ring by the requesting (missing) cluster. Note that even though all clusters in the system snoop on the probe, the time for each cluster to snoop does not add up to the miss latency because a probe message does not wait for the result of the snoop, but is immediately forwarded to the next cluster. In other words, the snooping time overlaps with the probe round-trip time. Thus, the total miss latency is roughly the time for a probe message to traverse the whole ring plus the time to fetch a block and insert it in the ring. This is true because as soon as the probe passes through the cluster with the valid copy, the block is fetched and the block message is sent. We call the time for a probe to complete a round-trip through the ring a *cycle*, and we use it as the basic unit to compare latencies, since the time to fetch a block and insert it in the ring is independent of the protocol used.

It is interesting to note that the miss latency, in the snooping protocol, is independent of the position of the cluster with the valid copy, which is why we claim that the slotted ring is an uniform access interconnection under this protocol, similar to a shared bus. This feature is not present in the other two protocols.

In a directory scheme [14] the miss latency depends on the global state of the missing block. If the block is not dirty, the latency is equal to one cycle, since in this case the home cluster provides a valid copy of the block. However, if the block is dirty, the position of the dirty cluster in relation to the home cluster significantly affects the miss latency. If

the dirty cluster is in the path from the home cluster to the requesting cluster, the miss will be satisfied in one cycle (but with two directory lookups), otherwise it will take two cycles for the missing block to reach the requesting cluster.

In the SCI protocol, the home cluster satisfies a miss only if the block is not currently being cached, i.e., the sharing list is empty. In this case the miss latency is one cycle. If the block is being cached, the head of the list is responsible for providing a copy of the block to the missing cluster, and then the latency of satisfying the miss depends on the relative position of the cluster at the head of the sharing list. If the cluster at the head of the list is in the path from the requesting cluster to the home cluster, the latency will be two cycles.

Invalidation Latency

Invalidation messages in the snooping protocol also require only one cycle, unless a given cluster rejects the invalidation probe (by setting the ns-bit). We will not consider this possibility since such rejections would compromise the performance of the other protocols as well.

In the directory protocol, invalidations require more than one cycle if the clusters to be invalidated are not in the path from the home node to the requesting node, but no more than three cycles are required in any situation.

Invalidations in the SCI protocol may require *0* to *P* cycles (where *P* is the number of clusters in the ring). If the cluster that generates the invalidation is the only one in the sharing list, no message is generated and the write operation takes place with no further delay. However, if there are *N* clusters in the sharing list, the invalidation may involve *N-1* cycles if the order in which the clusters appear in the sharing list is exactly inverted with respect to their order in the ring direction.

4.4 Discussion

The above analysis shows that the proposed cache coherence protocol outperforms the centralized directory protocol in terms of latency of coherence accesses. The only situation in which the SCI protocol may outperform the snooping protocol is when a cluster attempts to write to a RS block and it has the only cached copy of that block. In this case the snooping protocol sends an invalidation because the cluster does not know that it has the only cached copy and that there are no other copies to invalidate. In the SCI protocol, if the cluster at the head of the list sees itself as the only one in the sharing list, it modifies the block without delay. However, in all remaining situations, the SCI protocol exhibits much longer latencies than the snooping protocol, which we believe would lead to worse average performance figures.

It is also quite straightforward to extend our snooping protocol in order to avoid invalidation messages when a cluster has a private RS copy. This can be accomplished by including a *Read-Exclusive* cache state, as it has been done in the Illinois protocol [15]. We chose to omit the *Read-Exclusive* state from the previous snooping protocol description for the sake of simplicity, but its inclusion is a minor modification in the overall protocol structure.

The snooping protocol is also likely to generate less traffic than the other protocols considered, due to the fact that probes will never travel more than once through the ring (under conflict free transactions). There is no waste of bandwidth in broadcasting miss or invalidation probes in this architecture.

By arranging probes into frames and interleaving the dual-directory we hope to be able to avoid probe rejections completely, so that the ns-bit can be removed from the design.

To give an estimate of the processing power of the Express Ring with our coherence protocol consider a configuration with 16 clusters (32 processors) and a 200 MHz pipeline clock rate (the slotted ring has 48 stages which accommodates 6 frames). On the average, a miss occupies a probe slot for a full ring traversal and a block slot for half a ring traversal. Based on this, we can say that two misses can be satisfied as a frame makes a full ring traversal, and the expression for the maximum system miss rate that can be supported by the Express Ring under these assumptions is

$$\text{Maximum Miss Rate} = 6/(48*tc) = 25 \text{ M misses/sec.},$$

Where *tc* is the ring clock cycle (5 ns). For cache hit rates of 98%, and an average memory access per instruction rate of 1.2, such an Express Ring is able to support a peak performance of

$$25x10^6/(1.2*0.02) = 1.04 \text{ GIPS}$$

The latency of a probe request in such a system is less than 250 ns, which is very competitive with the latency of current interboard buses. A configuration with 2 pipeline stages per cluster and a 500 MHz clock could support 64 clusters (128 processors) with a probe round-trip latency under 300 ns.

5. Final Remarks

The objective of this study was to demonstrate that cache coherence can be efficiently maintained in a slotted ring architecture, by means of a relatively simple cache coherence protocol which is based on the classic bus snooping protocol. We believe that the Express Ring protocol demonstrates that point, and is an attractive alternative to maintain coherence in a shared-memory multiprocessor. The proposed protocol not only outperforms directory-based protocols and the SCI protocol, but is also less complex than both alternatives, in a ring-based

architecture.

Another important characteristic of our protocol is that, regardless of the relative positions of clusters in the ring, all ring transactions have the same latency, making the slotted ring a uniform access media. As a result, at the programmer level the Express Ring behaves as a very fast bus-connected multiprocessor, which makes it appropriate for general purpose computing.

The main limitation of the Express Ring architecture is that latency of accesses increases linearly with the number of clusters in the ring. Because of this, configurations with hundreds of clusters would experience very large delays to satisfy coherence requests. To be able to deal with that, we plan to experiment with latency tolerant techniques such as lockup-free caches [5] and delayed consistency [6]. Such techniques allow computation to be overlapped with communication, thus reducing the penalty for ring transactions. We are also interested in investigating how our protocol can be extended to allow configurations that include multiple Express Rings connected together in various manners. The Aquarius Multi-Multi protocol [2] may be an interesting scheme to adopt.

6. Acknowledgments

The authors wish to thank Dr. Kai Hwang for his support and Dr. Alvin Despain for valuable discussions about implementation issues. Mr. Pong Fong also helped us with some insight in the behavior of the SCI protocol.

7. References

[1] L. Barroso and M. Dubois, A Snooping Cache Coherence Protocol for a Ring Connected Multiprocessor, USC Tech. Report CENG 91-03, January 1991.

[2] M. Carlton and A. Despain, "Multiple-Bus Shared Memory System", IEEE Computer, *V*ol. 23, No. 6, June 1990, pp. 80-83.

[3] D. Chaiken, et al., "Directory-Based Cache Coherence in Large-Scale Multiprocessors", IEEE Computer, Vol. 23, No. 6, June 1990, pp. 49-59.

[4] D. Del Corso, M. Kirrman, and J. Nicoud, Microcomputer Buses and Links, Academic Press, 1986.

[5] M. Dubois and C. Scheurich, "Lockup-Free Caches in High-Performance Multiprocessors", The Journal of Parallel and Distributed Computing, January 1991, pp. 25-36.

[6] M. Dubois et al., Delayed Consistency and Its Effects On The Miss Rate of Parallel Programs, USC Technical Report, CENG 91-14, April 1991.

[7] M. Ferrante, "CYBERPLUS and MAP V Interprocessor Communications for Parallel and Array Processor Systems", Multiprocessors and Array Processors, W. J. Karplus editor, The Society for Computer Simulations, 1987, pp. 45-54.

[8] J. Goodman, "Using Cache Memory to Reduce Processor/Memory Traffic", Proc. of the 10th Int. Symp. on Computer Architecture, June 1983, pp. 124-131.

[9] R. Halstead Jr. et al., "Concert: Design of a Multiprocessor Development System", Proc. of the 13th Int. Symp. on Computer Architecture, June 1986, pp. 40-48.

[10] A. Hooper, R. Needham, "The Cambridge Fast Ring Networking System," IEEE Trans. on Computers, Vol. 37, No. 10, October 1988, pp. 1214-1224.

[11] D. Farber and K. Larson, "The System Architecture of the Distributed Computer System - the Communication System", Symp. on Computer Networks, Polytechnic Institute of Brooklyn, April 1972.

[12] D. James, "SCI (Scalable Coherent Interface) Cache Coherence", Cache and Interconnect Architectures In Multiprocessors, M. Dubois and S. Thakkar editors, Kluwer Academic Publishers, Massachusetts, 1990, pp. 189-208.

[13] R. Katz et al., "Implementing a Cache Consistency Protocol", Proc. of the 12th Int. Symp. on Computer Architecture, June 1985, pp. 276-283.

[14] D. Lenoski et al., "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor", Proc. of the 17th Int. Symp. on Computer Architecture, June 1990, pp. 148-160.

[15] M. Papamarcos and J. Patel, "A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories", Proc. of the 11th Int. Symp. on Computer Architecture, New York, 1986, pp. 414-423.

[16] J. Pierce, "How Far Can Data Loops Go?", IEEE Trans. on Communications, Vol COM-20, June 1972, pp. 527-530.

[17] SCI (Scalable Coherent Interface): An Overview, IEEE P1596: Part I, doc171-i, Draft 0.59, February 1990.

[18] P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors", IEEE Computer, Vol. 23, No. 6, June 1990, pp. 12-25.