A SPEED Cache Coherence Protocol For An Optical Multi-Access Interconnect Architecture

Joon-Ho Ha and Timothy M. Pinkston

CENG Technical Report 95-08

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4482

April 1995

A *SPEED* Cache Coherence Protocol for an Optical Multi-Access Interconnect Architecture

Timothy M. Pinkston Joon-Ho Ha

Electrical Engineering-Systems Department
University of Southern California
3740 McClintock Avenue, EEB-208
Los Angeles, CA 90089-2562
{tpink@charity.usc.edu, jha@truth.usc.edu}
{http://www.usc.edu/dept/ceng/faculty.html/pinkston/home.html}

Abstract

This paper presents a low overhead, high performance cache coherence protocol designed to exploit high bandwidth point-to-point and broadcast features of optics. *SPEED* integrates the virtues of snoopy-based schemes and directory-based schemes into one efficient protocol. Directory-assist is used exclusively for read traffic to eliminate unnecessary broadcasts while snoopy-assist is used exclusively for write and synchronization traffic to reduce directory overhead and synchronization complexities. The proposed protocol has the potential to increase performance as a result of its global independence between read and write operations, concurrency in channel access, reduced contention, and efficient broadcast of coherence and synchronization events.

1 Introduction

Caches play a critical role in reducing interconnect traffic and latency in shared memory multiprocessors (SMMPs) by maintaining local copies of shared data. An important factor that determines the performance of cached SMMPs is the cache coherence scheme. Hardware schemes for maintaining coherence of locally cached copies broadly fall into two categories: snoopy-based and directory-based schemes. These schemes depend very much on the underlying interconnect architecture which, traditionally, has been based on electronic technology.

Snoopy-based schemes implemented electronically, although simple, do not scale well mainly due to 1) limited bus bandwidth compounded by impedance matching problems with the electrical medium and 2) the snooping mechanism of each node becoming saturated with broadcast of all operations, many of which are irrelevant to most nodes. The scalability of directory-based schemes does not suffer directly from the limitations of the electronic medium but, rather, from the directory overhead needed to record those caches that have a copy of various memory blocks. Synchronization events (essential primitives in SMMPs) are also very complex and costly to implement with non-broadcast electrical point-to-point interconnects such as those generally used for directory-based cache coherence schemes.

Recent advances in optical technology has opened new opportunities for SMMP interconnect and cache coherence design to offset some of the problems associated with conventional approaches [1, 2, 3]. For instance, wavelength division multiplexing (WDM) over high-bandwidth optical fiber can result in speed-matched multiple channels for use as multiple point-to-point links or multiple optical buses, each of which can support a greater bandwidth than its electrical counterpart [1, 2, 4, 5]. WDM-based optical interconnects which transform aggregate bandwidth into multiple high bandwidth channels should do so in such a way as to result in efficient channel utilization. Efficient channel utilization can be achieved by balancing the communication load among the various optical channels.

This is done in a cache coherent SMMP by balancing the cache traffic among the optical channels – specifically, the coherence (write) and non-coherence (read) traffic.

This paper presents a cache coherence protocol designed for an optical multi-access interconnect architecture for SMMPs. Our cache coherence protocol maintains consistency across all nodes while allowing concurrent access to all channels. The remainder of this paper is organized as follows. The next section provides some background on the problem and compares our approach with other proposed approaches. Section 3 gives an overview of the proposed multi-access interconnect architecture. Section 4 presents a formal definition of the critical race problem which can occur in our scheme and discusses ways of detecting and resolving critical races. Section 5 describes a write-invalidate version of our proposed cache coherence protocol. Section 6 evaluates our design approach and identifies potential problems. Finally, we give conclusions and future work in Section 7.

2 Background

2.1 Motivation

Our optical multi-access interconnect architecture (referred to as *OMIA*) allows us to integrate the virtues of snoopy-based schemes and directory-based schemes into one efficient cache coherence protocol we call *SPEED* (Snoopy Protocol Enhanced and Extended with Directory). *SPEED* uses snoopy-assist over a broadcast channel for coherence operations only (e.g., invalidate or update actions for write-hits) and directory-assist over fully-connected channels for memory-block-request (MBR) operations (e.g., block transfer actions due to read-misses and write-misses). The motivation behind our approach is twofold. First, coherence operations are less frequent than MBR operations [6, 7]; therefore, the number of broadcasts should be only a fraction of total number of references over the network. Second, because coherence operations implicitly require synchronization, they can be performed more efficiently over the high-bandwidth shared channel provided by *OMIA*; hence, writes are performed on a separate broadcast channel as if they were

done on a shared bus. MBRs are directed onto a separate set of high-bandwidth point-to-point channels also provided by *OMIA*. Because only MBR operations need directory assistance for point-to-point communication to the owner of the requested block, the directory keeps track only of the owner (e.g., a cache or main memory) of each memory block in our scheme. This keeps the directory overhead of our scheme low in comparison to other proposed directory schemes [8, 9].

Our coherence protocol allows full concurrency between coherence and MBR operations targeted for the same memory block. This represents a drastic departure from conventional approaches which, in essence, serialize these operations either at the bus or at the directory, thus disallowing concurrency. This is because concurrency of this type can result in races. Races, if not properly handled, could lead not only to inconsistency among caches but also to other problems such as ambiguity in block ownership. Latency tolerating techniques such as pre-fetching and multi-threading which allow multiple outstanding MBRs could further exacerbate the problem. However, disallowing such concurrency could result in poor effective channel utilization, particularly for multi-link networks (e.g., k-ary n-cubes).

SPEED allows full concurrency by forcing each node to handle any potential problems (such as critical races) autonomously. Additional states are used in the protocol to cover the transient periods of cache/memory blocks being operated upon so that any races which may occur during a transient period are captured. The appropriate action can then be taken by nodes autonomously. What makes this possible is the atomic broadcast of coherence operations supported by *OMIA* via a separate high-bandwidth broadcast channel.

Hence, *SPEED* is designed with two goals in mind. The first is to maximize scalability by minimizing the use of centralized/global data structures, broadcasts and controls to only those cases where they are absolutely required. The second is to resolve as opposed to avoid problems due to races in order to boost channel utilization. Although *SPEED* is based on *OMIA*, we believe our proposed design provides a general framework for efficient use of optical technology to solving certain cache coherence problems (and communication problems in general) in SMMPs.

2.2 Related Work

Several SMMP systems based on multiplicity of electrical channels (links) have been proposed recently. We first consider snoopy-based systems. Mudge, et al. [10], proposed a time-multiplexed multiple bus scheme with centralized arbitration to accommodate several hundred processors. Their scheme primarily focused on increasing processor-memory bandwidth and not on dealing with associated cache coherence problems such as races. Wilson [11] proposed a hierarchical bus/cache architecture where multiple buses and caches are arranged into several levels of hierarchy. The cache protocol is an extended version of snoopy cache schemes for a single level shared bus, where some kind of serialization is used within the hierarchy to prohibit races such as acknowledgments or centralized control. The Wisconsin Multicube [12] uses a grid of buses to build a large-scale SMMP which employs a snoopy cache scheme that serializes potential races using a type of dimension-ordered routing on the grid of buses. The Multi-Multi system [13], which is also based on a grid of buses, uses a snoopy cache scheme on individual buses and a directory-based scheme among buses. In this system, the directories and the buses serve as serialization points.

The Alewife [14] and DASH [15] represent two experimental SMMPs employing directory-based cache schemes. The LimitLESS [16] cache protocol in Alewife combines hardware and software solutions in which each directory entry has a limited number of pointers to deal with the directory growth problem [9]. This protocol uses the directory as a serialization point for competing operations to the same memory block. In contrast to Alewife, DASH uses a full-map directory scheme for clusters and does not provide a unique point of serialization for shared objects. Instead it handles out-of-order and race conditions as exceptions. The lack of common serialization points result in more freedom in moving data around (e.g., direct cache forwarding). Our protocol follows in like manner.

What is common in all of the above electronic-based schemes, except DASH, is that they use some *global* serialization points to order simultaneous multiple operations occuring on multiple paths. This method is highly restrictive and inefficient in that it subjects every memory operation to some type of global serialization regardless of the actual need

for it. This could hamper efficient communication and load balancing if the serialization points are not well distributed and if the actual need for serialization is low. Our proposed protocol does not serialize operations globally. Coherence and MBR operations, although performed globally, are serialized locally only, if necessary. The benefit of local serialization is twofold in our protocol. First, the latency due to global serialization between coherence and MBR operations is totally eliminated. In contrast to global serialization, local serialization becomes more efficient as the frequency of actual required serializations decreases. Second, as coherence and MBR operations become globally independent, these operations can use channels independently and, hence, more efficiently.

The efficient use of multiple WDM-based optical channels for cached SMMPs has been studied by other researchers. Ghose, et al. [5, 17], proposed a WDM-based optical bus called OPTIMUL to explore the benefits of concurrent operation of multiple channels for both shared memory and message-passing multiprocessors. OPTIMUL for shared memory employs a snoopy *write-update* scheme to take advantage of the high bandwidth of optical links. This scheme allows multiple requests to be transmitted simultaneously on multiple channels but, like the previous schemes mentioned, globally serializes operations at the main memory to avoid race problems. In other work, Dowd, et al. [18], studied the interaction between different channel access controls and cache coherence protocols for SMMP. Their scheme is characterized by communication load balancing achieved by time multiplexed use of channels among nodes. Although this study involves a snoopy cache scheme in which coherence and MBR operations use separate channels, it does not address potential race problems. Hence, our work seems to stand alone in its approach to supporting and resolving problems dealing with cache coherency for multi-access interconnect architectures based on optics.

3 Overview of *OMIA*

Our optical multi-access interconnect architecture (*OMIA*) uses multiple high bandwidth channels of wavelength multiplexed optical fiber in such a way to support well, on a common medium, both broadcasting and point-to-point communication with reasonable scalability. This is achieved by partitioning the channels into two classes; one is used for broadcasting, and the other is used for unique point-to-point communication using the *self-routing* property of optics (e.g., wavelength recognition) [1].

Two channels are allocated for broadcasting in the system: the *control* channel and the *broadcast* channel. The *control* channel is shared by all the nodes in the system and is used for distributed arbitration of all other channels through a type of reservation scheme [18, 19]. A status table (data structure) resident in each node's optical network interface keeps track of channel reservation information which is broadcasted on the *control* channel. The *broadcast* channel is shared by all the processing nodes for broadcast of global events such as coherence operations and synchronization. All other channels, called *home* channels, are allocated for point-to-point communication between nodes. Every node is able to make a point-to-point connection to every other node by transmitting on the unique *home* channel of the destination node.

In *OMIA*, processing nodes can transmit data on all available channels but can receive data on only three different channels: the shared *broadcast* channel λ_b, the shared *control* channel λ_c, and its unique *home* channel λ_i. Memory nodes (which can be distributed at the processing nodes) are similar except that they cannot transmit on the *broadcast* channel. This significantly reduces *broadcast* channel contention. The optical network interface of each node handles data reception and traffic-control on multiple receiving channels and channel arbitration for data transmission. A primary function of the optical network interface is to receive MBR and coherence operations that might occur concurrently (even for the same memory block) and deliver them in order to the cache controller. Figure 1 shows a conceptual block diagram of the optical network interface for processing nodes.

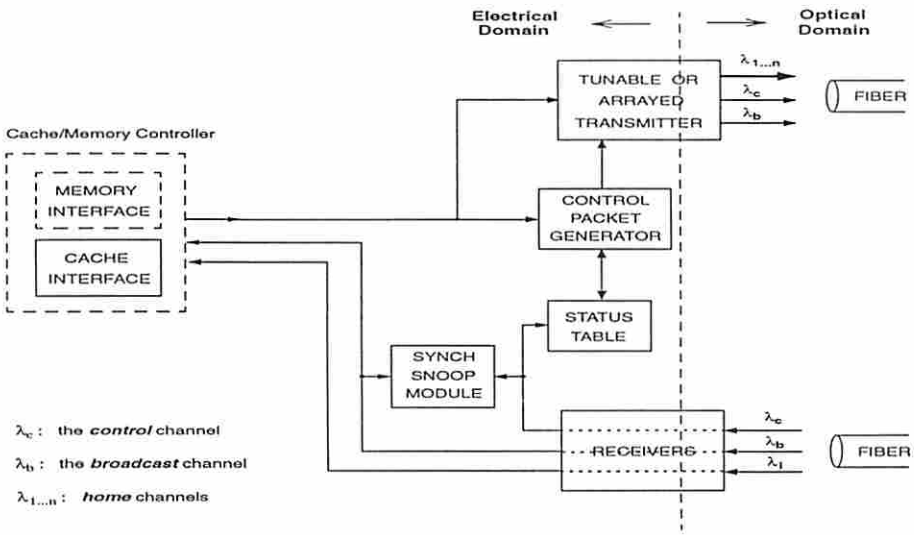

Figure 1: Block diagram of optical network interface at nodes.

OMIA can be implemented using optical fibers, optical star couplers, tunable or arrayed optical sources and fixed-wavelength optical receivers. WDM is used to transform the high bandwidth potential of optical fiber, which is estimated to be in 30 THz range [1], into multiple channels distinguished by wavelength. These channels are distributed to nodes using an optical star coupler [20]. Tunable transmitters [21] or an array of transmitters [22] fixed at different wavelengths can be used to implement the various channels. Fixed optical receivers [23] (three are required per node) can be used to selectively filter out received data on the *broadcast*, *control*, and unique *home* channels. Figure 2 shows a conceptual implementation of *OMIA* based on guided optics.

Scalability of *OMIA* largely depends on whether it can provide sufficient bandwidth and fan-in/fan-out with an increasing number of processors. Although optics has very high bandwidth and reasonable fanning capabilities, there are fundamental limitations. The bandwidth of the *broadcast* and *control* channels, which are shared among all nodes, can limit scalability to hundreds of nodes. The fan-in/fan-out capability of the star-coupler (due to optical power budget limitations [24] can limit scalability of *broadcast* and *home* channels to a few hundred nodes [20]. The fan-in of the fiber for the array of light sources

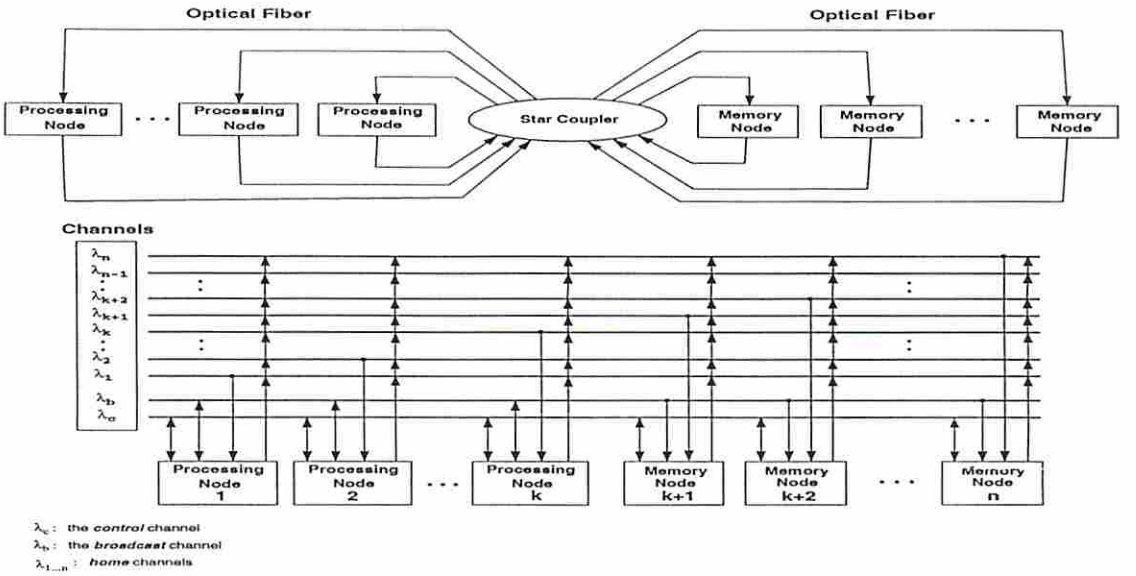

Figure 2: Conceptual implementation of proposed *OMIA* system.

or the tunability of a tunable source can also limit scalability of the *home* channels to a little less than a hundred nodes [21].

We suggest a number of ways to overcome many of these limitations. Bandwidth and fan-in/fan-out can be increased by employing a *multi-channel-broadcast* scheme in which a set of channels (implemented by duplicating hardware) is used for broadcasting. Here, the cost of implementing more channels using multiple star-couplers would now be the only limit to scalability. Alternatively, time multiplexing *home* channels among a group or cluster of nodes can increase scalability (similar to the GLORI Strategy [25]). Because the bandwidth demand for *home* channels is expected to be lower than that for the *broadcast* channels, this is a viable solution.

In summary, *OMIA* supports multiple transmissions on multiple channels over a common optical medium. A channel allocation policy is enforced that results in an interconnect architecture which has the broadcast/synchronization capability of a bus for coherence operations and the connectivity of a fully-connected network for non-coherence operations. As proposed, *OMIA* has moderate scalability.

4 The Critical Race Problem

In order for there to be a race between a coherence operation and a MBR operation, the two have to be concurrent. This is not possible in single-channel systems such as shared buses but is possible in multi-channel systems such as *OMIA* (and k-ary n-cube networks in general). Races may or may not affect the correctness of program execution, depending on whether they are critical or not. In this section, we first identify potential problems associated with races and present our solution for detecting and resolving these potential race problems.

4.1 Formal Problem Description

We formally define terms and assumptions used to prove an assertion about critical races.

Definition 1 (Race) *A* race *occurs between two operations pending to be performed on a given object if they overlap in time. Operation* **A** wins *a race with* **B** *if* **A** *is performed on the object before* **B** *is performed on that object.*

Definition 2 (Critical Period) *The* critical period *of a MBR operation is the time span over which the copy of a memory block being transferred in response to the MBR cannot be affected by racing coherence operations.*

Definition 3 (Critical Race) *A race between a MBR operation due to a cache miss and a coherence operation is said to be a* critical race *if the coherence operation completes during the critical period of the MBR operation.*

Definition 4 (Cache Inconsistency) *Cache inconsistency is said to occur if the contents of a given memory block which resides in multiple caches in the valid state is not identical across those caches.*

Let us make the following assumptions.

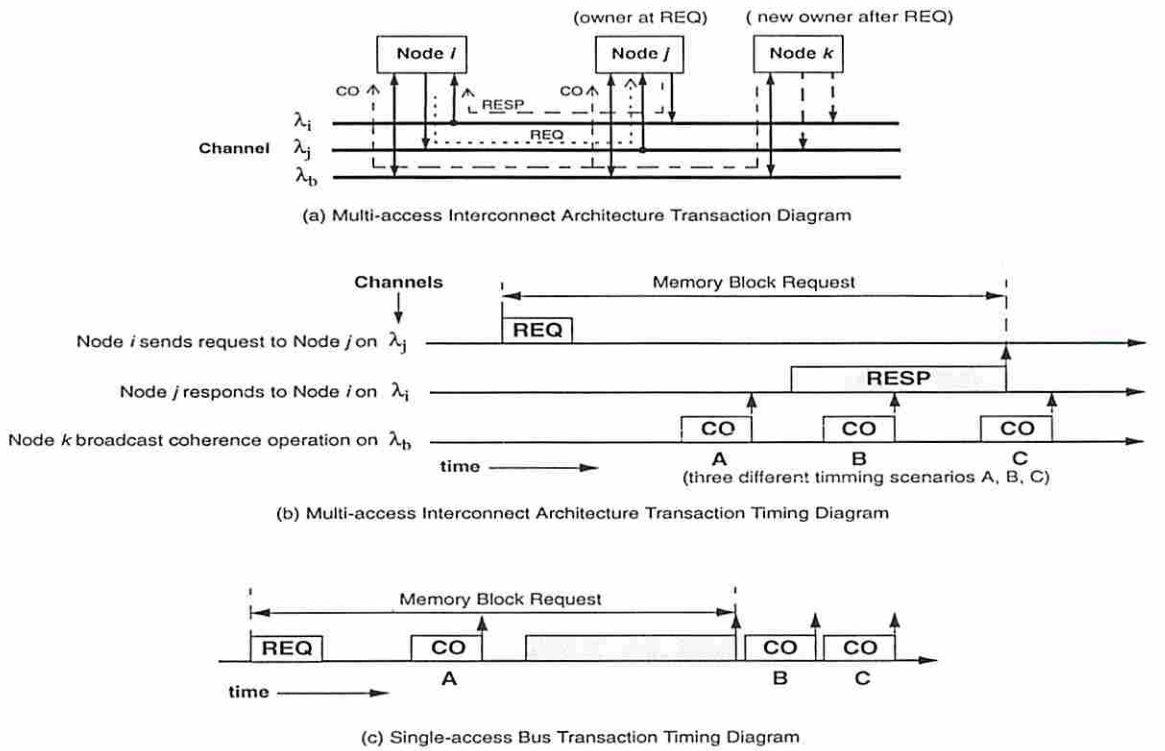

Figure 3: An illustration of possible race scenarios.

Assumption 1 *A pending memory block request is considered performed immediately after the RESP phase is completed* (as indicated by the up arrow in Figure 3).

That is, MBR operations can be (and are assumed to be in *OMIA*) split-phase transactions which involve the phase of sending a request for a memory block (REQ) and the phase of receiving a response to the request (RESP). REQ represents the time to initiate a MBR. RESP represents the indivisible portion of the phase of receiving a response, during which time the block location being filled is in a transient state. The data being received during this phase is essentially *locked-in* and is not affected by any intervening coherence operation (CO). The time span of RESP depends on block size and channel bandwidth.

Assumption 2 *A pending write which causes a coherence operation is considered performed at the time that the coherence operation is completed* (as indicated by the up arrow in Figure 3).

That is, COs are caused only by write transactions. Each CO represents the time to broadcast a coherence operation, which takes effect upon completion of the CO.

Assumption 3 *Memory operations are based on the notion of* block-ownership *in which each memory block has a unique owner (e.g., cache or main memory).*

That is, there is a unique owner of a requested memory block, and it is responsible for responding to requests. Ownership of a block is *acquired* by a cache on the event of writing to it and is re-acquired by memory on write-back. This assumption is specific to *SPEED* but can be true of other schemes.

Assumption 4 *If, while in the process of responding to a MBR, the current owner of a memory block loses ownership before the critical period of that transfer begins, that owner (now the previous owner) either negatively acknowledges the requestor or forwards the MBR to the new owner.*

That is, orderly transfer of block ownership is maintained, and MBRs are not lost during transference of ownership.

We can now make the following assertion about critical races.

Assertion 1 *A critical race between a coherence operation and a MBR operation on a cache block object results in inconsistency among caches.*

Proof 1 The proof follows from considering all case scenarios as given in Figure 3. Figure 3(b) illustrates the timing for the three possible scenarios of a coherence operation racing with a MBR operation for the optical multi-access interconnect network shown in Figure 3(a). For race scenario **A**, CO takes effect before RESP begins. From Definition 3, this race is not critical. The current owner is forced to relinquish ownership and respond to the pending MBR by negatively acknowledging it or forwarding it to the new owner (Assumptions 3 and 4). In either case, the RESP will come from the legitimate owner and will be up-to-date, resulting in consistency among caches. In race scenario **C**, CO loses the

race with the MBR operation and, therefore, takes effect on the block which has already been transferred (Assumptions 1 and 2). Here, too, the race is not critical and coherence is maintained. In race scenario **B**, however, CO wins the race with the RESP (Assumption 2) and takes effect during the critical period, making this a critical race (Definition 3). From Definition 2, the critical period of a MBR operation minimally corresponds to the RESP phase but could be longer, depending on the details of the architecture, e.g., buffering, interrupt handling, etc. CO fails to invalidate or update the block being received while the MBR is in the RESP phase (Assumption 1). Consequently, the received block, which is now stale, remains valid once the MBR is performed (after the RESP phase is completed). From Definition 4, we see that caches are inconsistent. This results in the race being critical.

Evidently, the RESP phase of MBR operations in competition with coherence operations is the primary source of critical races as defined. As illustrated in Figure 3(c), a race between a RESP and a CO cannot occur on a single-channel shared bus but can occur on a multi-channel network such as *OMIA* as proved above. Because *OMIA* allows critical races to occur, our cache coherence protocol must protect against them to ensure consistency.

4.2 Critical Race Detection and Resolution

Schemes that deal with critical races broadly fall into two categories: critical race avoidance and critical race resolution schemes. We first examine avoidance schemes.

Critical races can be avoided by simply not allowing coherence and MBR operations to be performed concurrently as is done inherently on a shared-bus by virtue of its topology. This simple but highly restrictive scheme would severely limit the efficiency of channel utilization. Also, the parallelism inherent in those operations cannot be exploited with this technique. To improve upon this, mechanisms can be used to detect conditions for critical races; here, serialization of operations to avoid critical races would occur only when these conditions exist. Although concurrency is preserved, detection of critical race

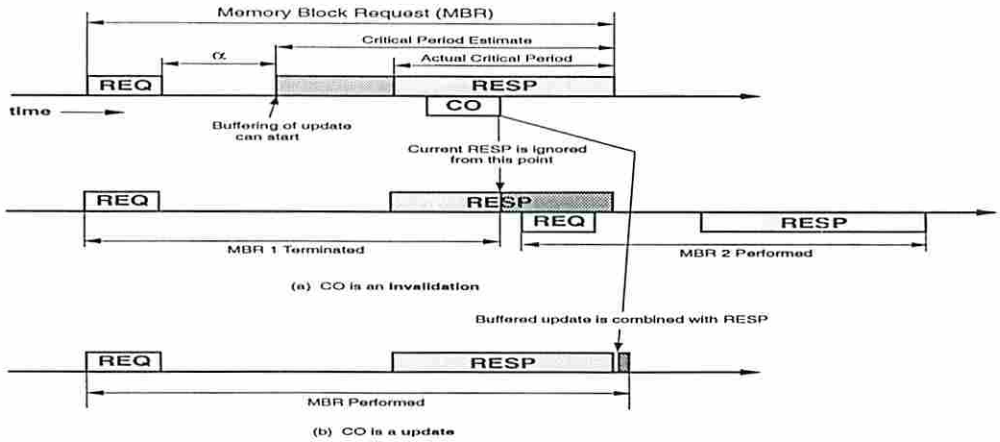

Figure 4: An illustration of critical race resolution in *SPEED*.

conditions would require global knowledge of every MBR being issued. The benefit of this selective serialization could quickly be offset by the overhead of making MBR issuance globally known. Consequently, an imbalance of communication loads among channels is unavoidable, resulting in inefficient channel utilization.

An alternative to race avoidance is race resolution. Our scheme does not avoid critical races; instead, distributed mechanisms are provided to detect and resolve them as they occur using only locally available information. *SPEED* resolves potential cache inconsistency resulting from critical races by forcing a would-be inconsistent copy to remain invalid or to become consistent with other copies before becoming valid.

This can be achieved for write-invalidate schemes by ignoring the corrupted MBR that loses the critical race as shown in Figure 4 (a). This results in an extra MBR. Hence, this violates Assumption 1, making the MBR responsive to the CO and effectively serializing the two operations. For write-update schemes, the updates involved in critical races can be buffered by the node that made the competing MBR and, then, combined with the received block upon completion of an MBR's RESP phase. In this case, all intervening updates involved in critical races are combined with the received block before the MBR is considered to be performed. Hence, this violates Assumption 2, making the CO take effect after the RESP and extending the MBR to include the CO, again effectively serializing the

two operations. Therefore, for both the invalidate and the update versions of *SPEED*, the result is the same as if these operations were serialized on a shared bus.

Critical races can be detected by detecting intervening coherence operations during the critical period of a MBR operation. Thus, the key issue in detecting critical races is determining the critical period. One way of determining the critical period is to estimate it given certain system parameters (e.g., network latency, memory access, etc.) This is similar in concept to the *fixed-time-response* mechanism used in the HP 9000 Model T500 for the processor-memory bus [26]. The beginning of a critical period is estimated to be the time when REQ is completed plus α, which is the minimum latency[1] for a responding node to put a copy of the requested block in the un-interruptible state for transmission. Hence, the complexity of precisely determining the critical period can be avoided by using estimates. However, precise determination of the critical period eliminates undue critical races. Although both result in correct program execution, undue critical races result in extra MBR operations for write-invalidate schemes.

In summary, our scheme does not avoid critical races by disallowing them. Instead, it safely resolves their potential harmful effects. Moreover, critical races are detected locally and resolved in a distributed manner at each node. Since critical races are allowed, there is no restriction on channel usage due to critical races. Consequently, the parallelism inherent in operations that potentially can be involved in critical races is better exploited with our technique. Hence, better channel utilization is expected as communication loads are more balanced among channels without undue restrictions.

5 The Proposed Cache Coherence Protocol

In this section, we describe the design of *I-SPEED*, an invalidate version of *SPEED*. The description of *U-SPEED*, an update version of *SPEED*, is beyond the scope of this paper. Many features of existing snoopy and directory-based protocols are usable in *OMIA* with-

[1]The minimum latency results from memory/cache and network access without contention.

out any modification. Hence, our design focuses only on those aspects that are directly concerned with the integration of snoopy and directory techniques into one robust protocol and those issues related to the implementation of our critical race resolution scheme. The main advantage of our protocol is the elimination of unnecessary broadcasts and increased channel utilization, both of which improve scalability through load distribution and decentralization.

5.1 Main Features of *I-SPEED*

In our write-invalidate protocol, *I-SPEED*, snooping and broadcast are used only for invalidations due to writes and events that require global synchronization. MBR operations are performed with minimal directory assistance using point-to-point communication.

Our protocol addresses two major issues. One deals with the general problem of maintaining consistency among cache blocks and main memory in the presence of critical races. The other deals with maintaining orderly transfer of blocks and block ownership among caches and main memory in the presence of critical races. The same mechanism which is used to resolve critical races is used to solve the problems in block transfer and ownership.

5.1.1 The Owner Directory

In our protocol, each memory block has a unique owner (cache or main memory) which is responsible for responding to MBRs and writing-back cache blocks on replacement at all times. One of the key features of *SPEED* is that caches take the responsibility of block ownership transfer and, thus, no centralized control for the transfer of block ownership is necessary. Caches acquire block ownership by invalidating copies before writing to their own copy. Main memory, which is the initial owner of all the blocks, re-acquires ownership only on write-back. Owners of all the memory blocks are listed in a data structure, called the *owner-directory*, in main memory. Invalidation broadcasts are used not only to invalidate blocks but also to update the owner-directory. MBRs are sent first to main memory which, if not the owner, forwards MBRs to the owner as specified by the owner-directory.

The owner directory has much smaller overhead than other directory schemes. *SPEED* requires only $\log N$ bits per memory block to identify an owner, where N is the number of nodes able to cache the block, Full-map directory schemes require N bits per node, which certainly is less scalable. *SPEED*'s owner directory overhead matches the minimum hardware overhead required by limited pointer schemes such as LimitLESS [9] which, for the same overhead cost, can store only one pointer. What's more, MBR operations in *SPEED* involve only two nodes and do not require synchronization. This greatly simplifies the role that the memory controller must play in performing MBR operations.

5.1.2 The Pseudo-block

As MBR operations and invalidations can overlap, it is necessary to represent the state of a block for which a MBR operation is being performed in relation to overlapping invalidations. In our protocol, we achieve this by labeling a block for which a MBR is pending as a *pseudo-block*. A pseudo-block is used to cover the transition period of a block during the MBR operation. It is logically identified by the *pending block address*. A pseudo-block comes into existence during a MBR operation primarily to detect and appropriately represent overlapping operations.

Critical races can be covered[2] by using a pseudo-block during the critical period of a MBR operation in the requesting node. Once a pseudo-block is set up, any racing invalidations which win the race can invalidate the pseudo-block. In this case, an invalid pseudo-block indicates that a critical race has occured to the MBR.

5.1.3 Handling Write-misses

Write-misses can be handled in two ways. In one scheme, the request for a block and the invalidation of other copies are combined on a write-miss into a single MBR during the REQ phase (e.g., REQ+INV) which is broadcasted to all nodes. This is similar to write-invalidate protocols presented in [27]. In this scheme, a pseudo-block is set up at the

[2]In other words, the system watches for critical races and resolves them if they occur.

end of the REQ+INV phase (request combined with invalidation) of the combined MBR and is used to detect and serialize locally any overlapping MBRs that could occur during the period from the invalidation broadcast to the RESP completion of the combined MBR. Overlapping MBRs are not responded to until the RESP phase of the pending MBR and write is performed. This is because block ownership (in pseudo form) is transferred at the end of the REQ+INV phase, but the block itself is not transferred until the RESP phase. Hence, critical races between MBR operations due to overlapping write-misses are resolved as Assumption 2 is violated.

This scheme offers two benefits. First, the number of channel accesses for a write-miss operation can be reduced at least by one because REQ is combined with INV. Second, the latency of sending requests is reduced because requests are broadcasted to the owner without having to refer to the owner-directory, which would be an extra channel hop.

A second scheme for handling write-misses is to perform write-misses as a sequence of a read-miss (MBR) followed by a write-hit (INV). Few, if any, write-invalidate protocols able to implement this due to network constraints. In this scheme, write-misses per se do not exist. Pseudo-blocks are used to cover critical races for the MBRs of both regular read-misses and read-misses resulting from write-misses. With this scheme, at least one more channel access is performed for the REQ phase of a MBR operation. Also, the latency of sending a request is longer due to the access to the owner-directory.

For both schemes, broadcasting is used only once; however, the first scheme trades off simplicity for a reduced number of channel accesses and latency. We examine both schemes in greater detail in the next section.

5.2 The *I-SPEED* Protocol

I-SPEED associates four different states with each cache and memory block; *clean*, *exclusive*, *shared*, and *invalid*. Two bits are associated with each block to indicate its state. This protocol implements a *single-owned-copy* mechanism which allows only one copy to be in the *exclusive* or *shared* state, states which identify the owner. To enforce this, cache-forwarded

copies due to read-misses are received as *clean* copies. This protocol defines pseudo-blocks to be in one of three pseudo-states; *pseudo-clean*, *pseudo-owned*, and *pseudo-owned-invalid*. Two bits, called *pseudo-block-bits*, are assigned to each pseudo-block to indicate a pseudo-state. The number of pseudo-blocks required depends on the number of outstanding MBRs allowed.

We summarize these states as follows:

- *clean*: a cache block is *clean* if it is consistent with other cache copies and main memory.

- *exclusive*: a cache block is *exclusive* if it has been written into. An *exclusive* copy is the *only* valid copy in the system and, by definition, is dirty. The cache or memory module with an *exclusive* copy is the owner of the block.

- *shared*: an *exclusive* block becomes *shared* if the block has been forwarded. Ownership remains unchanged. Note that the cache to which the copy has been forwarded marks the block as *clean*, even though it may be dirty from memory's point of view.

- *invalid*: a cache block is *invalid* if it has not been brought into the cache or has been invalidated. A memory block is *invalid* if it has been invalidated by a cache which becomes the new owner.

- *pseudo-clean*: a pseudo-block due to a read-miss is *pseudo-clean*. The *pseudo-clean* state is used to cover the critical period of a MBR.

- *pseudo-owned*: a pseudo-block due to a write-miss is *pseudo-owned*. The *pseudo-owned* state is used to cover the period in which the owner does not have the actual block (i.e., due to the REQ+INV phase of a MBR).

- *pseudo-owned-invalid*: a *pseudo-owned* block becomes *pseudo-owned-invalid* if an invalidation to the block has occurred.

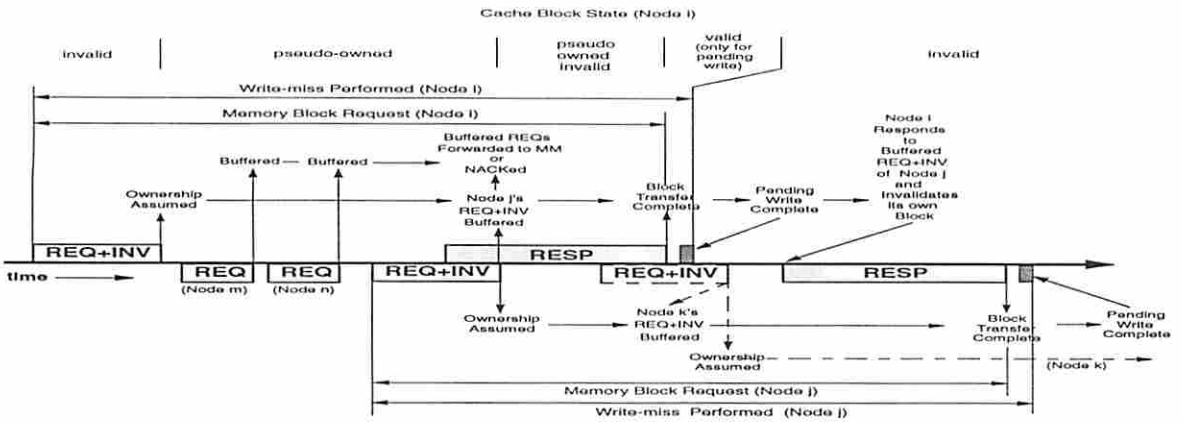

Figure 5: Example of *I-SPEED Combined.*

The first four states are based on the concepts in [27, 28]. We introduce the use of *pseudo-states* to accomplish two things in particular. One is to detect critical races. The other is to ensure that block transfer and ownership is done in proper order. The *pseudo-clean* state is added to indicate the critical period of a MBR operation due to a read-miss. Any intervening invalidation to a *pseudo-clean* block is interpreted as a critical race. This results in the discarding of the block being transferred and an immediate issuance of another MBR to service the same read-miss. Thus, critical race events for read-misses have the same effect as a read-miss. Below, we consider the two schemes for handling write-misses.

5.2.1 *I-SPEED Combined*

In what follows, we describe *I-SPEED* for the first write-miss scheme, where the requestor combines its invalidation with its MBR during the REQ phase of the operation (e.g., REQ+INV). We refer to this as *I-SPEED Combined.*

The *pseudo-owned* state is used to represent the logical transfer of ownership to the cache issuing a REQ+INV on a write-miss. As illustrated in Figure 5, nodes i and j assume ownership of the same block being requested once each REQ+INV completes. As the block itself has not yet been transfered, a pseudo-ownership is used to maintain the

order in which real ownership should be granted once the operation completes. This is done by detecting and buffering the first overlapping REQ+INV following the current REQ+INV. The *pseudo-owned-invalid* state indicates that an overlapping REQ+INV to a *pseudo-owned* block has occurred. This state forces a cache to forward the transfered block and ownership in response to the buffered REQ+INV immediately after the pending write is performed and to invalidate its own copy. In the figure, Node i buffers the REQ+INV from Node j and responds to it after the pending write is performed. Note that Node i need not buffer the REQ+INV from Node k as Node j automatically takes responsibility for it. Hence, buffering serializes overlapping write-misses so that the requested block and ownership is transfered to caches in the order in which they issue REQ+INVs. Coherency is maintained.

Overlapping read-misses can be a problem in this scheme. Since the owner does not actually have the block during the *pseudo-owned* state, it has to either buffer or negatively acknowledge REQs of overlapping read-misses. If buffering is used, REQs are first buffered and then responded to after the pending write-miss completes, similar to REQ+INV. If a REQ+INV occurs (transitioning the pseudo-block to the *pseudo-owned-invalid* state) while some REQs are buffered, the buffered REQs end up in the old owner. This situation can be handled in two ways. REQs can be forwarded back to main memory by the old owner to be redirected to the new owner or they can be negatively acknowledged. In Figure 5, two REQs from nodes m and n are buffered in Node i. As the REQ+INV from Node j overlaps with Node i's MBR, the buffered REQs can be forwarded back to main memory or negatively acknowledged. If REQs are buffered, the buffer size required per node must be as large as the number of processing nodes, but the actual number typically used should be much less. Negative acknowledgments can, at the very least, double the latency of overlapping read-misses but is much simpler to implement as no REQ buffers are needed.

Figure 6(a) illustrates the state transition diagram of cache blocks and pseudo-blocks. Figure 6(b) shows the pseudo-block mechanism for a node where a single outstanding MBR is allowed. Race problems for multiple outstanding requests can be handled by increasing

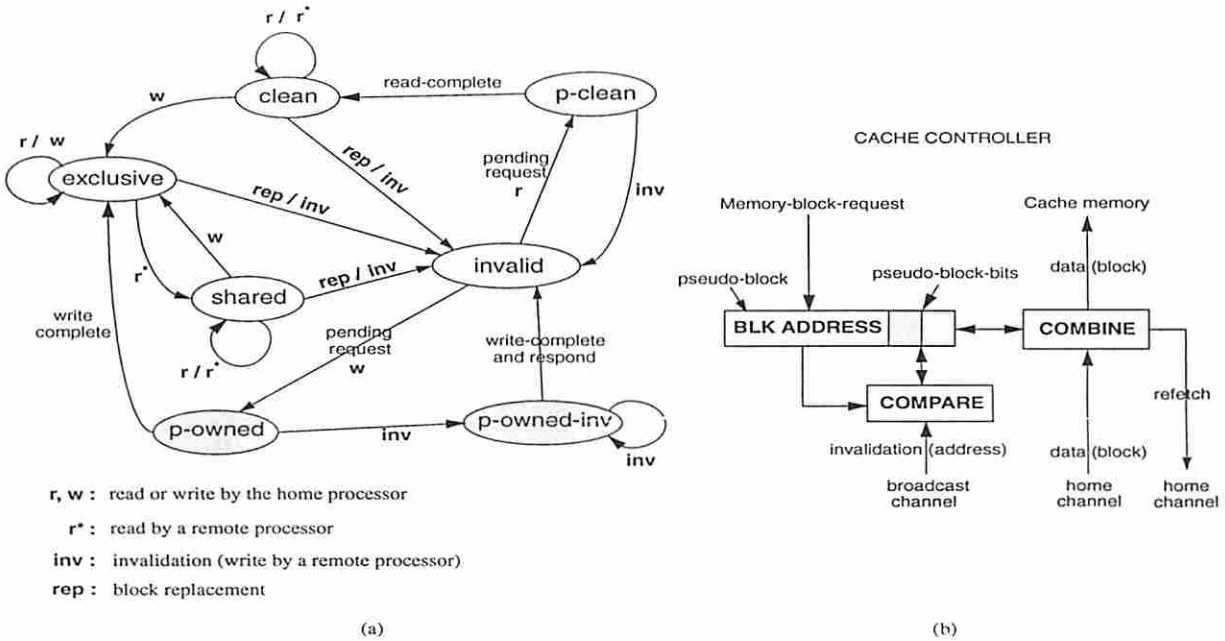

Figure 6: (a) State transition diagram and (b) mechanism for pseudo block update in cache controller.

the number of pseudo-blocks per cache. One pseudo-block for each outstanding MBR suffices to deal with the race problem. In the case of a read-miss, a pseudo-block is set up and set to the *pseudo-clean* state; an intervening invalidation to the pseudo-block causes the pseudo-block and the block being transferred to be dismissed immediately and the cache block to remain invalid. In the case of a write-miss, a pseudo-block is set to the *pseudo-owned* state. On the first overlapping REQ+INV, it is buffered and the pseudo-block is set to *pseudo-owned-invalid* state. The *pseudo-owned-invalid* state forces a cache to respond to the buffered REQ+INV after the write-miss is performed. Not shown in the figure is the buffer to store the intervening REQ+INV (only the first needs to be buffered).

5.2.2 *I-SPEED Combo*

In our second protocol, a write-miss is performed as the sequential combination of a read-miss (MBR) and a write-hit (INV) as illustrated in Figure 7. We refer to this as *I-SPEED Combo*. In effect, *I-SPEED Combo* eliminates the need for a pseudo-block to

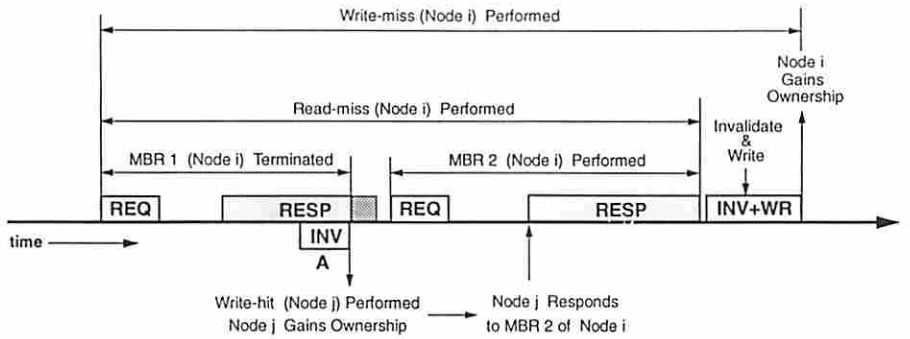

Figure 7: Example of *I-SPEED Combo*.

cover write-misses as MBRs now are solely due to read-misses. This is the only difference between the two variants on *I-SPEED*.

Figure 7 shows an intervening write-hit in Node j (INV), resulting in a critical race. Figure 8(a) illustrates the state transition diagram of cache blocks and pseudo-blocks for *I-SPEED Combo*. The mechanism to deal with race problems in Figure 8(b) has one less bit to mark the state of a pseudo-block because this mechanism does not distinguish between read and write misses. Moreover, there is no need to buffer REQs.

5.3 Cache Procedures

Now, we define the procedures of read, write, and block replacement by a cache for the proposed *I-SPEED* cache protocol on *OMIA*.

- Read-hit: Read. No state transition is involved on read-hits.
- Read-miss: On a read-miss, a MBR is sent on the *home* channel of the corresponding memory node. The memory node as the owner responds on the *home* channel of the requesting node. If the memory node is not the owner, it forwards the request to the owner as specified by the owner-directory on the *home* channel of the owner.
- Write-hit: On a write-hit to a *clean* or *shared* copy, an invalidation is broadcasted on the *broadcast* channel, and then the pending write is performed. A write-hit to an

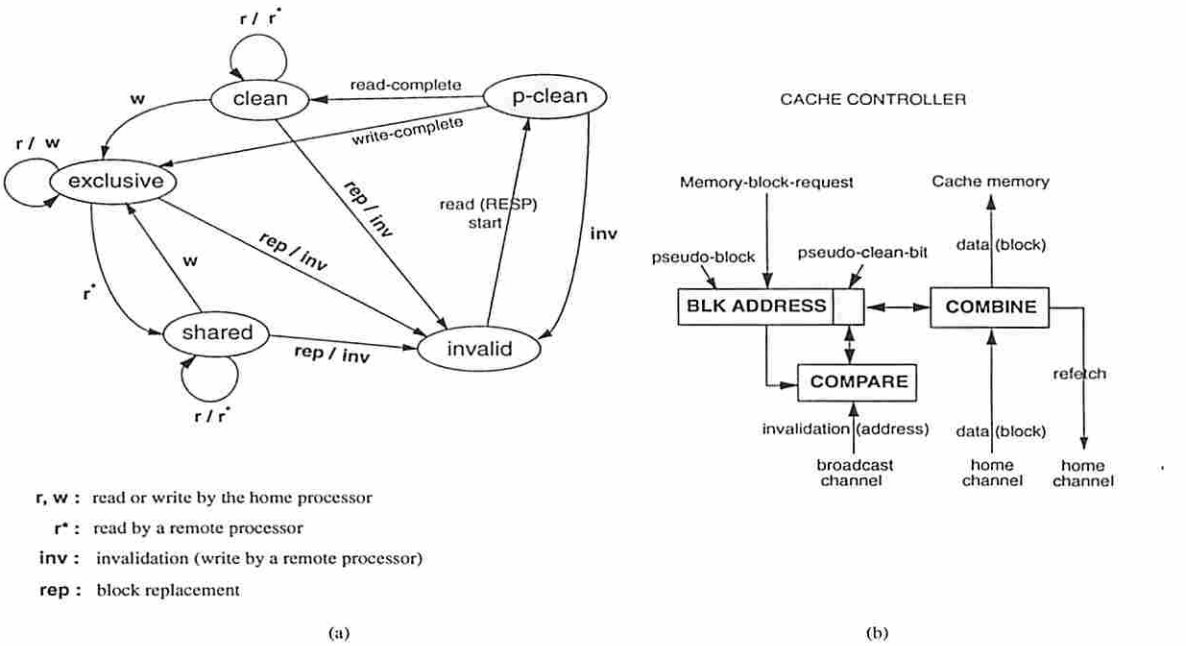

Figure 8: (a) State transition diagram and (b) mechanism for pseudo block in cache controller.

exclusive copy does not initiate an invalidation. A cache block becomes *exclusive* on a write-hit.

- Write-miss: As described previously, a write-miss can be handled in two ways as follows:

  1. *I-SPEED Combined*: A MBR combined with an invalidation (REQ+INV) is broadcasted on the *broadcast* channel. The owner responds on the *home* channel of the requesting node and, at the same time, invalidates its own copy. The pending write is performed on reception of the response (RESP).
  2. *I-SPEED Combo*: A read-miss and a write-hit (as described above) are performed successively.

- Replace: On replacement of an *exclusive* or *shared* block, a write-back is performed on the *home* channel of the corresponding memory node.

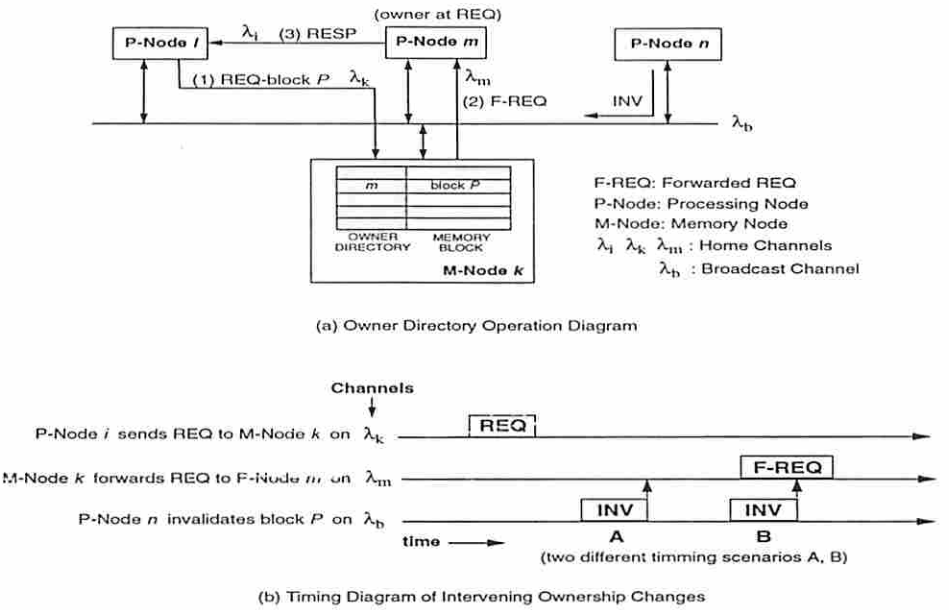

Figure 9: Diagram of owner-directory operation for a a read-miss MBR.

Figure 9(a) illustrates how a read-miss is performed using the owner-directory. In the figure, Processor-Node i sends a request for block P to the corresponding Memory-Node k. As it is not presently the owner, Memory-Node k forwards the request to the current owner, Processor-Node m, as given by the info in the owner-directory. Processor-Node m responds to Processor-Node i. All the operations above are performed on the *home* channel of each node, thus relieving the broadcast channel of excessive read traffic. Because REQs are sent on *home* channels, invalidations overlapping with the forwarding of REQs can lead to critical races, as the case with Processor-Node n.

In Figure 9(b), we show two scenarios of overlapping invalidation. In race scenario **A**, the INV takes effect before the F-REQ takes place. Thus, Memory-Node k is safely able to send the F-REQ to the new owner, Processor-Node n. In race scenario **B**, however, the INV takes effect in the middle of the F-REQ. Because of that, the F-REQ ends up in a node which is not the owner. As in the case of Figure 5, we use one of the prescribed methods to deal with this situation. The F-REQ can be forwarded back to main memory

so that it can be re-forwarded to the new owner or it can be negatively acknowledged to notify the requesting node.

5.4 Write-Back

Care has to be taken in dealing with write-backs because they involve un-requested transfer of blocks and block ownership to main memory. In both variants of *I-SPEED*, a write-back is viewed as a special case of a MBR operation – that due to a miss *in main memory*. It is different from ordinary MBR operations due to misses in caches in two ways. First, this MBR operation does not have a REQ from main memory. Only a RESP phase in which the actual write-back is performed exists. Second, it involves the transfer of block ownership to main memory. The absence of a REQ forces the node initiating a write-back to broadcast a message signaling the write-back to main memory before the actual write-back. This message is used to set up a pseudo-block in main memory to cover the write-back. Once the pseudo-block is set up, a write-back is performed like a read-miss. The timing of ownership transfer is set to the end of a write-back (RESP). An intervening invalidation results in a critical race. In such a case, the block being written back is invalidated immediately as discussed previously. The write-back is considered performed regardless of the outcome of races, if one exists. The key idea behind this mechanism is that the main memory acts like caches when a write-back is performed.

Table 1 summarizes the potential problems caused by overlapping operations for all combinations of cache and memory operations and lists our proposed solutions. The first two columns specify pending operations and their phases. The next two columns list those overlapping operations and their phases that can transpire simultaneously on other channels. Finally, the last two columns list the potential problems resulting from these overlapping operations and our proposed solutions.

Pending OP	Phase	Overlapping OP	Phase	Problem in Pending OP	Solution
RH (read-hit)	READ	RH	READ	none	
		RM	all	none	
		WH	all	none	
		WM	all	none	
		WB	all	none	
RM (read-miss)	all	RH	all	none	
	all	RM	all	none	
	after REQ	WH	INV	REQ at invalid node	NACK REQ or forward REQ to MM
	RESP			critical race	pseudo-block
	after REQ	WM	INV	REQ at invalid node	NACK REQ or forward REQ to MM
	RESP			(A) critical race	pseudo-block
				(B) critical race	pseudo-block
	REQ	WB	RESP	none	
WH (write-hit)	INV	RH	n/a	n/a	
	INV	RM	all	none	
	INV	WH	n/a	n/a	
	INV	WM	all	(A) none	
				(B) none	
	INV	WB	all	none	
WM (write-miss) Combined	after REQ+INV	RH	n/a	n/a	
	after REQ+INV	RM	REQ	REQ to pseudo-owner	buffer REQ or NACK REQ
	after REQ+INV	WH	n/a	n/a	
	after REQ+INV	WM	REQ+INV	REQ to pseudo-owner	buffer REQ
	REQ+INV	WB	RESP	none	
WM (write-miss) Combo	INV	RH	n/a	n/a	
	INV	RM	all	none	
	INV	WH	n/a	n/a	
	INV	WM	REQ+RESP	none	
	INV	WB	RESP	none	
WB (write-back)	RESP	RH	READ	none	
	RESP	RM	REQ	none	
	RESP	WH	INV	critical race	pseudo-block
	RESP	WM	(A) REQ+INV	critical race	pseudo-block
			(B) INV	critical race	pseudo-block
	RESP	WB	n/a	n/a	

OP: Operation
MM: Main memory
NACK: Negatively acknowledge
all: Whole MBR operation
n/a: The combination cannot occur

Table 1: Summary of problems caused by overlapping OPs and solutions.

6 Discussion

Our design has the potential to increase performance in two important ways: reduce write latency and reduce read latency. Performance is increased because of the global independence between read and write operations, concurrency in channel access, reduced contention, and broadcast of synchronization events including invalidations.

Write latency: We may attribute two aspects of our design to the reduction of write latency. For one, by broadcasting coherence operations (due to writes), invalidations are *single-hop* network operations, resulting in overall lower write latency. Moreover, synchronization following an invalidation comes for free. Second, by not broadcasting MBR operations (due to read-misses) the broadcast channel does not become saturated, leaving more broadcast bandwidth for those operations which truly need it – writes. This also reduces write latency. Although write latency can be hidden with weak memory consistency models [29, 30], the benefit of this reduced write latency becomes greater for stronger consistency models requiring sequential consistency [31].

Read latency: Read latency plays more of a critical role in determining system performance than write latency because processors are usually stalled on read-misses, regardless of the memory consistency model. Three aspects of our design contribute to reduced read latency. For one, there is a point-to-point channel from each node to every other node dedicated to MBR operations (for read-misses). Our protocol maximizes channel utilization by allowing concurrency of operations on these channels and on the broadcast channel, serializing them locally only if required (i.e., critical races, memory contention). Second, there is minimal contention at the directory as the directory access latency is reduced. On a write, a directory location is "locked" only during the period of its contents being updated (recording change of ownership status) as opposed to the entire time of globally performing the invalidate typical of most directory schemes. (This would correspond to the period between the beginning of the REQ+INV and the beginning of the RESP.) Third, direct-cache-forwarding where the owner responds directly to the requesting node mini-

mizes the latency of reads as responses do not have to go through main memory every time.

We mention below a few problems that could arise in *SPEED* and some proposed solutions. Critical races and change of ownership in our protocol involves forwarding or negatively acknowledging (an re-issuing) MBRs. This could result in a form of livelock where the solution or combination of solutions is pathologically repeated for some time. In particular, critical races or change in ownership could be repeated continuously for MBRs from the same miss on a heavily shared block, causing undue delay in one or more nodes. We believe that such livelock events would be extremely rare. Nevertheless, a solution to livelock of this type is to have some sort of *time-out* mechanism. In the worst case, the delay would be long enough to cause a time-out which could trigger an error condition.

A second solution to livelock is to defer some invalidations, essentially back-outing of potential livelocks. Memory operations based on the concept of block ownership (such as in our scheme) provide a good opportunity for doing this. For instance, to have a repeated critical race for a second MBR owing to the initial critical race, two sources of racing invalidations must exist: the current owner and write-misses in other nodes. While the current owner is responding to the second MBR, it can prevent another critical race by deferring the issuance of its own invalidation and responses to the write-misses in other nodes. Consequently, by the time any invalidations occur, the second MBR operation would be completed safely, escaping another critical race. The problem of repeated forwarding of MBR is more complex to deal with. One generic solution would be to broadcast an MBR on the broadcast channel after a certain number of forwarding events have occured. This, too, eliminates livelock.

7 Conclusions and Future Work

SPEED is an integrated cache coherence protocol designed to exploit high bandwidth point-to-point and broadcast features provided by our proposed optical multi-access in-

terconnect architecture (*OMIA*). In *SPEED*, directory-assist is used for memory block requests (e.g., read-misses) to eliminate unnecessary broadcasts whereas snoopy-assist is used for coherence operations (e.g., writes) to reduce directory overhead and synchronization complexities. Given typical communication behavior of shared memory machines, we believe this notion of decoupling the allocation of read resources from write resources can result in balanced channel usage and reduced read/write latency.

A distinct advantage of *SPEED* is that it does not enforce global serialization of concurrent operations occuring on different channels. Hence, concurrent reads and writes to a given memory block (and certainly to different memory blocks) are independently performed globally to boost channel utilization. Potential critical race problems which may arise are resolved by *SPEED* with the use of a pseudo-block mechanism. This enforces a serialization of those operations competing in critical races *locally* at the node(s) requiring it so as to maintain consistency between caches. The logic required to implement the pseudo-block mechanism is very small. Existing cache controller logic can be used with the addition of a few bits and buffers. The pseudo-block mechanism of *SPEED* can be applied to other schemes where global independence of reads and writes has the potential to increase performance. Hence, *SPEED* achieves low overhead in comparison to other scalable cache schemes in that directory overhead is minimal, directory access latency is minimal, and pseudo-block implementation is nominal.

There are many areas of future research resulting from this work. More work research is needed in the verification of our *SPEED* protocol. Also, it would be interesting to measure the performance increase afforded by our critical race resolution scheme in comparison to avoidance schemes. An update-based version of *SPEED* (*U-SPEED*) can be designed that may make even better use of the high bandwidth potential of optics. Finally, guided-wave and perhaps free-space implementations of *OMIA* should be further explored to address feasibility and scalability issues.

References

[1] Charles A. Brackett. "Dense Wavelength Division Multiplexing Networks: Principles and Applications". *IEEE Journal on Selected Areas of Communications*, 8:948–964, August 1990.

[2] Biswanath Mukherjee. "WDM-Based Local Lightwave Networks Part I: Single-Hop Systems". *IEEE Network*, pages 12–27, May 1992.

[3] H. Scott Hinton, Tom J. Cloonan, JR. Frederick B. McCormick, Anthony L. Lentine, and Frank A.P. Tooley. "Free-Space Digital Optical Systems". *Proceedings of the IEEE*, 82(11):1632–1648, November 1994.

[4] U. Krackhardt, F. Sauer, W. Stork, and N. Streibl. "Concept for an Optical Bus-type Interconnection Network". *Applied Optics*, 31:1730–1734, April 1992.

[5] Kanad Ghose, R. Kym Horsell, and Nitin K. Singhvi. "Hybrid Multiprocessing Using WDM Optical Fiber Interconnections". In *Proceedings of the First International Workshop on Massively Parallel Processing Using Optical Interconnections*, April 1994.

[6] S. Eggers and R. Kats. "The Effect of Sharing on the Cache and Bus Performance of Parallel Programs". In *Proceedings of the 3rd International Conference on on Architectural Support for Programming Languages and Operating Systems*, pages 257–270, May 1989.

[7] Anoop Gupta and Wolf-Dietrich Weber. "Cache Invalidation Patterns in Shared-Memory Multiprocessors". *IEEE Transaction on Computers*, 41(7):794–810, July 1992.

[8] Per Stenström. "A Survey of Cache Coherence Schemes for Multiprocessors". *IEEE Computer*, 23(6):12–24, June 1990.

[9] A. Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. "An Evaluation of Directory Schemes for Cache Coherence". In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 280–289. IEEE Computer Society Press, 1988.

[10] T. N. Mudge, J. P. Hayes, and D. C. Winsor. "Multiple Bus Architectures". *IEEE Computer*, pages 42–48, June 1987.

[11] Andrew W. Wilson Jr. "Hierarchical Cache/bus Architecture for Shared Memory Multiprocessors". In *Proccedings of the 14th Annual International Symposium on Computer Architecture*, pages 244–252. IEEE Computer Society Press, 1987.

[12] J.R. Goodman and P.J. Woest. "The Wisconsin Multicube: A New Large-Scale Cache Coherent Multiprocessor". In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 422–431. IEEE Computer Society Press, 1988.

[13] Michael Carlton and Alvin Despain. "Aquarius Project". *IEEE Computer*, 23(6):80–83, June 1990.

[14] Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk Johnson, David Kranz, John Kubiatowicz, Beng-Hong Limand Ken Mackenzie, and Donald Yeung. "The MIT Alewife Machine: Architecture and Performance". In *Proceedings of the 22nd International Symposium on Computer Architecture*. To be appear, June 1995.

[15] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor". In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 148–159. IEEE Computer Society Press, 1990.

[16] David Chaiken, John Kubiatowicz, and Anant Agarwal. "LimitLESS Directories: A Scalable Cache Coherence Scheme". In *Proceedings of the 4th International Conference*

on Architectural Support for Programming Languages and Operating Systems, pages 224–234. ACM, 1991.

[17] Kanad Ghose, R. Kym Horsell, and Nitin Singhvi. "Hybrid Multiprocessing in OPTIMUL: A Multiprocessor for Distributed and Shared Memory Multiprocessing with WDM Optical Fiber Interconnections". In *Proceedings of the 1994 International Conference on Parallel Processing*, volume 1, pages 196–199. CRC Press Inc., August 1994.

[18] Patrick W. Dowd and John Chu. "Photonic Architectures for Distributed Shared Memory". In *Proceedings of the First International Workshop on Massively Parallel Processing Using Optical Interconnections*, pages 151–161. IEEE Computer Society Press, April 1994.

[19] Kalyani Bogineni and Patrick W. Dowd. "A Collisionless Multiple Access Protocol for a Wavelength Division Multiplexed Star-Coupled Configuration: Architecture and Performance Analysis". *Journal of Lightwave Technology*, 10(11):1688–1699, 1992.

[20] K. Kato et al. "Packaging of large-scale integrated-optic n*n star couplers". *IEEE Photonics Technology Letters*, 5(3):348–351, 1993.

[21] M. Kuznetsov et al. "Frequency tuning characteristics and WDM channel access of the semiconductor 3-branch Y3-laser". *IEEE Photonics Technology Letters*, 6(2):157–160, 1994.

[22] J.L. Jewell, Y.H. Lee, A. Scherer, S.L. McCall, N.A. Olsson, J.P. Harbison, and L.T. Florez. "Surface-emitting microlasers for photonic switching and interchip connections". *Optical Engineering*, 29(3):210–214, 1990.

[23] A.H. Gauck et at. "A transimpedance APD optical receiver operating at 10 Gbps". *IEEE Photonics Technology Letters*, 29(1):114–115, January 1993.

[24] Timothy M. Pinkston. "Design Considerations for Optical Interconnects in Parallel Computers". In *Proceedings of the First International Workshop on Massively Parallel Processing using Optical Interconnects"*, pages 306–322. IEEE Computer Society Press, April 1994.

[25] Timothy M. Pinkston and Joseph W. Goodman. "Design of an Optical Shared Bus-Hypercube Interconnect". *Applied Optics*, 3(8):1434–1443, March 1994.

[26] Thomas B. Alexander, Kenneth G. Robertson, Dean T. Lindsay, Donald L. Rogers, John R. Obermyer, John R. Keller, Keith Y. Oka, and Marlin M. Jones. "Corporate Business Servers: An Alternative to Mainframes for Business Computing". *Hewlett Packard Journal*, pages 8–30, June 1994.

[27] J. Archibald and J.-L. Baer. "Cache Coherence Protocols: Evaluation using a Multiprocessor Simulation Model". *ACM Trans. Comput. Sys.*, 4:273–298, November 1986.

[28] L. M. Censier and P. Feautrier. "A New Solution to Coherence Problems in Multicache Systems". *IEEE Transactions on Computers*, C-27:1112–1118, December 1978.

[29] M. Dubois, C. Scheurich, and F. Briggs. "Memory access buffering in multiprocessors". In *Proceedings of the 13th International Symposium on Computer Archi tecture*, pages 434–442. IEEE Computer Society Press, 1986.

[30] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. "Memory consistency and event ordering in scalable shared-memory multiprocessors". In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 15–26. IEEE Computer Society Press, 1990.

[31] Leslie Lamport. "How to make a multiprocessor computer that correctly executes multiprocessor programs". *IEEE Transactions on Computers*, C-28(9):241–248, September 1979.