SPEED DMON: Cache Coherence on an Optical Multi-channel Interconnect Architecture

Joon-Ho Ha and Timothy Mark Pinkston

CENG Technical Report 95-26

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

December 1995

SPEED DMON: Cache Coherence on an Optical Multi-channel Interconnect Architecture

Joon-Ho Ha and Timothy Mark Pinkston

CENG Technical Report 95-26

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

December 1995

SPEED DMON: Cache Coherence on an Optical Multi-channel Interconnect Architecture

Joon-Ho Ha Timothy Mark Pinkston

SMART Interconnects Group
EE-Systems Dept., University of Southern California
Los Angeles, CA 90089-2562
{jha@truth.usc.edu, tpink@charity.usc.edu}
{http://www.usc.edu/dept/ceng/pinkston/SMART.html}

Abstract

This paper presents a low overhead, high performance cache coherence protocol and interconnect scheme designed to exploit high bandwidth point-to-point and broadcast features of a decoupled multi-channel optical network. *SPEED DMON* integrates the virtues of snoopy-based schemes and directory-based schemes into one robust hybrid architecture. Directory-assist is used exclusively for decoupled 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 scheme has the potential to increase communication performance as a result of its global independence between read and write operations, full concurrency in channel access, reduced link contention, and efficient broadcast of coherence and synchronization events.

1 Introduction

Recent advancements in optical technology open new and exciting opportunities for exploring the design space of distributed shared-memory multiprocessors (DSMs). Wavelength-division multiplexing (WDM) over high-bandwidth optical fiber (tens of THz) offers speed-matched multiple channels that can support far greater bandwidth than electrical lines [1, 2]. Passive optical star couplers offer all-to-all interconnect connectability for point-to-point broadcast of multiple high-bandwidth WDM channels [3, 4]. High-speed (nanosecond) tunable optical transmitters and receivers offer additional degrees of interconnect switching and reconfigurability [5]. Since optics has the potential to exceed fundamental performance limitations of electronics in the communications domain, novel interconnect architectures based on this technology deserve serious consideration. One area which seems most promising is the development of hybrid cache coherence protocols and interconnect schemes which exploit the high bandwidth point-to-point and broadcast capability of optics.

Caches play a vital role in reducing DSM interconnect traffic and latency by maintaining coherent copies of shared data locally. An important factor that determines the performance of cached DSMs is the scheme by which caches are kept coherent. Snoopy schemes based on electronic buses suffer directly from 1) low bus bandwidth due to impedance matching problems [6] and 2) the snooping mechanism of each node being saturated with broadcasts of all operations, many of which are irrelevant to most nodes. Directory schemes based on non-broadcast electrical point-to-point interconnects suffer from excessive directory overhead and complex/costly implementation of synchronization events, essential primitives in DSMs. Hence, any hybrid cache protocol and interconnect architecture should aim to efficiently utilize the aggregate optical bandwidth provided by the multiple high-bandwidth channels. Channels are utilized efficiently by balancing the communication load among them—specifically, cache traffic consisting of coherence (write) and non-coherence (read-miss) traffic.

This paper presents our proposed *S*noopy *P*rotocol *E*nhanced and *E*xtended with *D*irectory (referred to as *SPEED*) which is targeted for our proposed *D*ecoupled *M*ulti-channel *O*ptical *N*etwork (referred to as *DMON*). *SPEED DMON* integrates the virtues of both snoopy and directory cache/interconnect schemes into one hybrid architecture. *SPEED* uses snoopy-assist over a high-bandwidth broadcast channel provided by *DMON* for coherence operations (COs)

only, e.g., invalidate or update actions for writes, and directory-assist over high-bandwidth fully-connected channels also provided by *DMON* for memory-block-request operations (MBRs), e.g., block transfer actions due to read-misses. The motivation behind *SPEED DMON*'s resource allocation is the following. First, COs are less frequent than MBRs [7, 8]; therefore, the number of broadcasts are only a fraction of the total number of references over the network. Second, because COs implicitly require synchronization, they are performed efficiently over the high-bandwidth shared channel provided by *DMON*. Third, because only MBRs need directory assistance for point-to-point communication to the owner of the requested block, *SPEED*'s directory keeps track of the owner of each memory block only, e.g., cache or main memory. This keeps the directory overhead of our scheme low in comparison to other proposed directory schemes [9, 10, 11]. Hence, *SPEED* maximizes scalability by minimizing the use of centralized/global data structures, broadcasts and controls to only those situations where they are absolutely required. Although *SPEED* is initially targeted for *DMON*, we believe this protocol design provides a general framework for cache coherence in other optically as well as non-optically interconnected DSMs.

The remainder of this paper is organized as follows. Section 2 gives an overview of the proposed decoupled multi-channel optical interconnect architecture. Section 3 describes the *SPEED* cache coherence protocol. Section 4 presents performance results. Section 5 discusses related work. Finally, conclusions and future research are presented in Section 6.

2 *DMON* Optical Multi-channel Interconnect Architecture

The proposed *DMON* optical multi-channel interconnect architecture uses multiple high-bandwidth channels of wavelength multiplexed optical fiber to support well, on a common medium, both broadcasting and point-to-point communication with moderate upward and downward scalability. This is achieved by partitioning the available channels into two classes using the *self-routing* property of optics (i.e., wavelength recognition) [1]. One class is used for *broadcasting* and mimics the functionality of a shared bus; the other is used for unique *point-to-point* communication and mimics the functionality of a fully-connected network. Two channels shared by all nodes in the system, the *control* channel and the *broadcast* channel, are allocated for broadcasting. The *control* channel is used for distributed arbitration of all other channels, e.g., through a reser-

vation scheme [12]. The *broadcast* channel is shared by all processing nodes for broadcast of global events such as coherence operations and synchronization. All other channels, called *home* channels, are allocated for direct communication between nodes: every node is able to make a point-to-point connection to any other node by transmitting on the unique *home* channel of that destination node. These channels are used only for memory block request operations.

DMON allows processing nodes to transmit data on all available channels but to receive data on only three specific channels: the shared *broadcast* channel λ_b, the shared *control* channel λ_c, and a unique *home* channel λ_i (specific to each processor). 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 channel arbitration and data reception on multiple receiving channels. A primary function of the optical network interface is to receive memory-block-request and coherence operations that might arrive concurrently (even for the same memory block) and deliver them in-order to the cache controller [13]. Figure 2 shows functional blocks of the optical network interface chip we are currently fabricating using 0.8 micron hybrid CMOS-SEED technology.

The specification of *home* channels as *unique receiving frequencies* is one distinctive feature of *DMON* that sets it apart from other proposed WDM-based optical multi-channel interconnect architectures [12, 14, 15]. This better reflects the basic communication behavior of DSMs as destination nodes typically have no control over the initiation of communication by source nodes. *DMON*'s destination *home* channel allocation keeps control at the source end, allowing the source to initiate communication once the unique *home* channel of the destination is free (the destination remains out of the control loop). The decoupling of channel resources based on reference type (i.e., reads allocated to *home* channels and writes allocated to the *broadcast* channel) is another distinctive feature of *DMON*. This design feature exploits expected DSM communication behavior as well, as explained in Section 2.1.

Guided optical implementations for *DMON* can be achieved using optical fibers, optical star couplers, tunable or arrayed optical transmitters and fixed-wavelength optical receivers, as illustrated in Figure 1. Multiple channels distinguished by wavelength can be distributed to nodes using a passive optical star coupler (which is based on broadcast). Tunable transmitters [5] or an array of transmitters [16] fixed at different wavelengths can be used to implement the various channels. Fixed optical receivers [17] (three are required per node) can be used to selectively

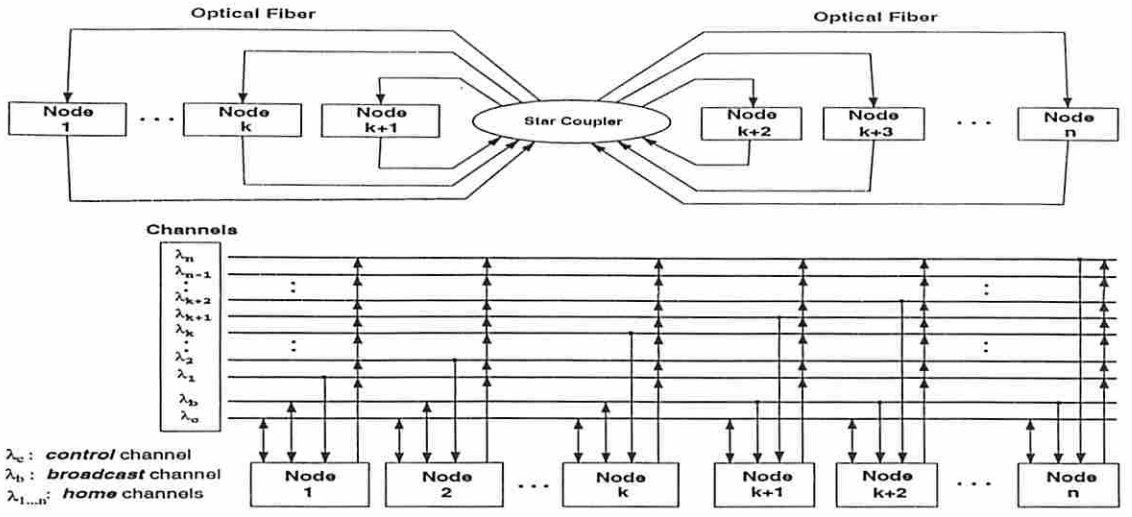

Figure 1: Conceptual implementation of the proposed *DMON*.

filter out unwanted channels that are inherently broadcasted by the star coupler to implement the *broadcast*, *control*, and unique *home* channels. Similar imaging components for modulating, splitting, and receiving photonic data are available for free-space *DMON* implementations.

Scalability of *DMON* 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 [18]) can limit scalability of *broadcast* and *home* channels to a few hundred nodes [3]. The fan-in of the fiber for the array of light sources or the tunability of a tunable source can also limit scalability of the *home* channels to approximately a hundred nodes [19]. However, these limitations may be overcome by the following. 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 (à la the GLORI Strategy [20]). As the bandwidth demand for *home* channels is expected to be lower than that for the *broadcast* channels, this is a viable solution.

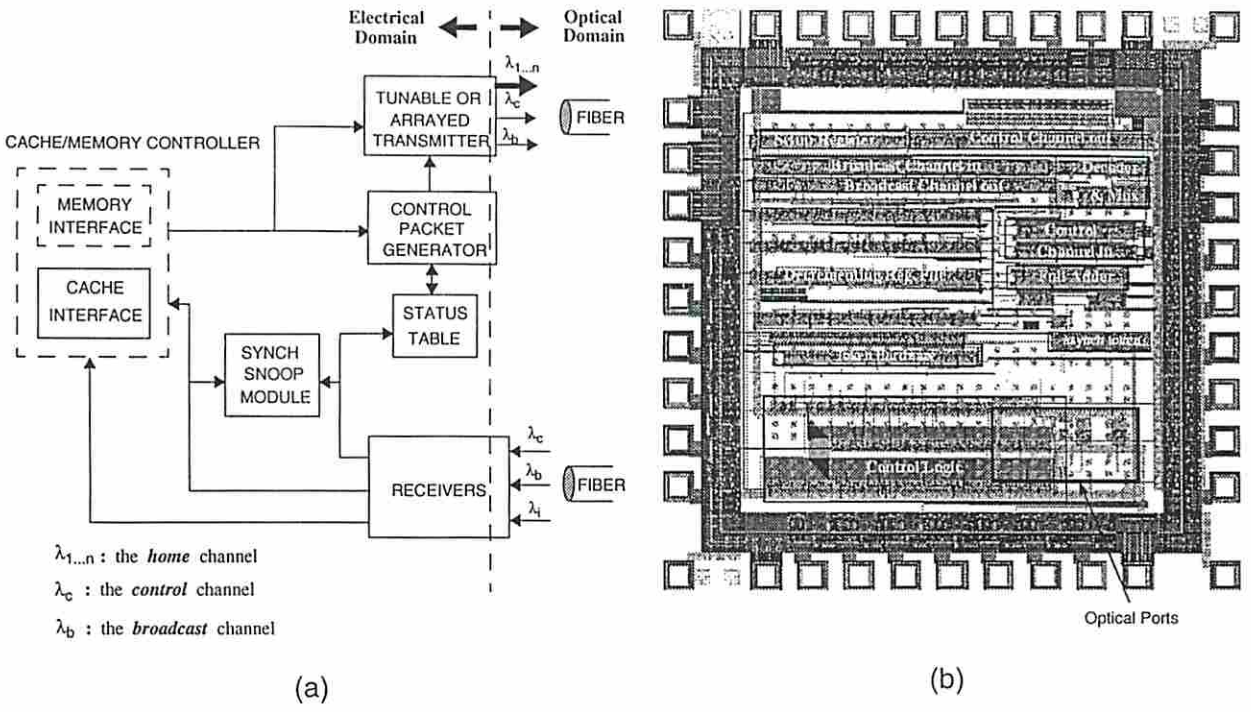

Figure 2: (a) Functional blocks of the optical network interface and (b) *DMON* interface chip based on hybrid CMOS-SEED integration.

2.1 Decoupled Reads and Writes

The communication traffic in cached DSMs primarily consists of memory block transfers (MBR operations) due to cache misses and coherence traffic (COs) due to shared writes[1]. *DMON* allocates communication resources for this mixture of traffic in a decoupled manner where COs are performed using the shared *broadcast* channel resource and MBRs are performed using the point-to-point *home* channel resources. Below we explain why this decoupled allocation is expected to provide balanced channel utilization.

Communication Requirements: CO and MBR operations differ in their communication requirements. COs typically require one-to-many communication and implicit synchronization whereas MBRs typically require one-to-one communication. Hence, COs are efficiently performed by multicasting via shared broadcast while MBRs are efficiently performed by point-to-point unicasting (shared broadcast is overkill).

Bandwidth Requirements: Many studies have shown that, for many parallel applications, writes requiring COs are less frequent than reads requiring MBR operations [7, 8, 9]. Our

[1]Shared writes are writes to memory locations accessible by two or more processors.

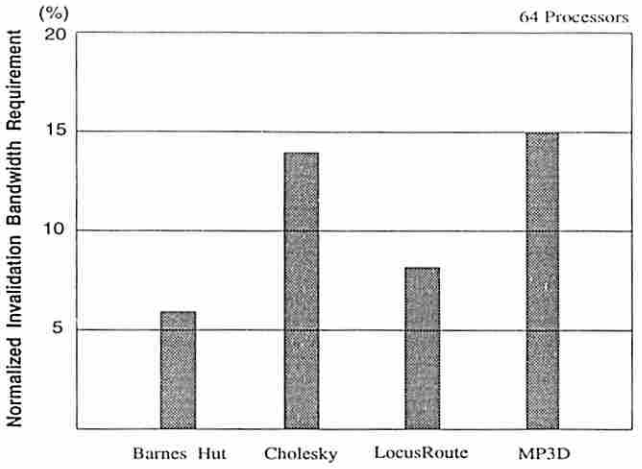

Figure 3: The aggregate bandwidth requirements of invalidation traffic normalized to total traffic.

characterization studies confirm this. Hence, with the decoupling of reads from the broadcast resource used for writes, a significant reduction of traffic on the *broadcast* channel can result for balanced communication load. In addition, each CO consumes less bandwidth than their counterpart MBR operations because the unit of transfer is typically much smaller (word-size vs. block-size). Therefore, *DMON*'s shared *broadcast* channel is likely to require less aggregate bandwidth than the point-to-point *home* channels as shown in Figure 3 and, therefore, not become a bottleneck given optics' high bandwidth capacity.

Scaling Requirements: Although *DMON*'s decoupling can significantly reduce broadcast traffic, the rate of growth of broadcast traffic with processor scaling needs to be addressed. If the broadcast network tends to saturated well before the point-to-point network, the proposed decoupling would be a bad idea. Our traffic characterization studies confirm other studies (see [8]) which show that coherence traffic does not explode with processor scaling for "well-written" parallel programs and coherence protocols which have "exclusive" states (i.e., COs are generated only for actively shared writes). Figure 4 shows that as the number of processors increases by a factor of eight CO traffic increases by at most only a factor of two. Given this trend, *DMON*'s shared *broadcast* channel is not expected to become a bottleneck through scaling for moderately sized systems.

2.2 Potential Benefits: Latency Reduction

The proposed decoupling has the potential to reduce read and write latency. By not broadcasting MBR operations, the *broadcast* channel contention is reduced, leaving more broadcast bandwidth

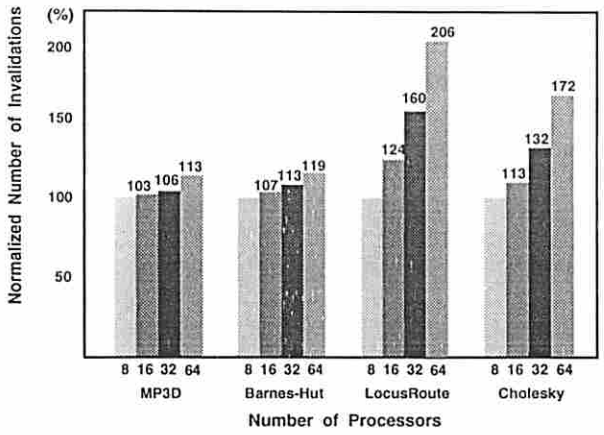

Figure 4: Increase in coherence traffic with system scaling (the number of broadcasts are normalized to an 8 processor system).

for those operations which truly need it – writes. (A similar reduction in point-to-point *home* channel contention is seen for reads.) Although write latency can be hidden with weaker memory consistency models [21, 22], this benefit of reduced write latency becomes greater for stricter memory consistency models [23]. Read latency is more critical than write latency as processors typically stall on read-misses, regardless of the memory consistency model. By allowing MBR operations performed on the dedicated point-to-point *home* channels to occur concurrently with COs on the shared *broadcast* channel, decoupling can further reduce read latency. Thus, latency due to contention and serialization of reads with writes in the network is totally eliminated. This provides greater ability to prioritize reads over writes (to minimize read latency) without being inhibited by inherent ordering which might otherwise occur within the network.

Maximum benefit from *DMON*'s decoupling of reads and writes is gained when the cache coherence protocol exploits this feature. By maintaining coherence using a snoopy-like mechanism, coherence transactions require just a single broadcast hop as compared to multiple point-to-point hops if maintained by a directory-like mechanism. This hop-count reduction results in lower overall write latency. Similarly, by maintaining block ownership information using a directory-like mechanism, reads are able to be serviced faster without needlessly saturating the snoop mechanism. This leads to the notion of a hybrid cache coherence scheme for *DMON* such as *SPEED* which uses snoopy-assist for COs and directory-assist for MBR operations. Before we present this hybrid protocol, we first discuss some potential cache coherence-related hazards that accompany read/write decoupling.

2.3 Potential Hazards: Critical Races

Decoupling as proposed makes cache coherence-related serialization between concurrent CO and MBR operations from two or more nodes to the same memory location a difficult task (if this becomes necessary). For example, in the absence of a shared bus or a lock-up directory which serves as a global serialization point for both reads and writes, the two operations transpiring concurrently and targeted for the same memory/cache block can end up racing with each other. This type of race may affect the correctness of program execution if not handled correctly. Below, we formally identify potential problems associated with such races between CO and MBR operations by defining *critical* and *non-critical* races. Following this, we discuss how the cache coherence protocol can be used to efficiently resolve these races locally as opposed to restricting global concurrency by altogether preventing them from occuring.

2.3.1 Formal Problem Description

We formally define terms and assumptions used to prove an assertion about critical races and cache consistency. The following discussion is applicable to both unified (atomic MBR transactions) as well as split-phase MBR transactions which involve the phase of sending a request for a memory block (REQ) followed non-atomically by the phase of receiving a response to the request (RESP). REQ represents the time to initiate an MBR. RESP represents the indivisible portion of the phase of receiving a response to the MBR.

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.*

In general, the *critical period* depends on implementation details such as the transmission time of memory blocks, the electro-optic conversion time, and the buffering time at the network interface, etc. During this period, the cache block state as controlled by the cache/memory controller is

locked-in to what it was just before the start of the critical period when the cache block is being filled.

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.*

Non-critical races are the races that are not *critical.*

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.* (This is a modification of the definition given in [24] now applicable in the context of cache contents.)

We make the following assumptions consistent with certain *DMON*-specific and invalidation-based cache design details.

Assumption 1 *A pending memory block request is considered performed immediately after the RESP phase completes* (as indicated by the up arrow in Figure 5).

The RESP phase represents the critical period of cache block fill at the receiving node. Therefore, the cache controller changes the cache block to a valid state only after the RESP phase is ended.

Assumption 2 *A pending write which causes a coherence operation is considered performed at the time that the coherence operation completes.* (as indicated by the up arrow in Figure 5).

That is, COs are considered to take effect and complete once the broadcast transmission is ended.

Assumption 3 *The cache controller is not responsive to COs targeted to blocks that are already in the invalid state.*

That is, COs take effect only on cache blocks that are not already invalid; otherwise, they are ignored by the cache controller.

Assumption 4 *A* write-back *policy is enforced such that memory operations are based on* block-ownership *in which each memory block has a unique owner (e.g., cache or main memory) at any given time.*

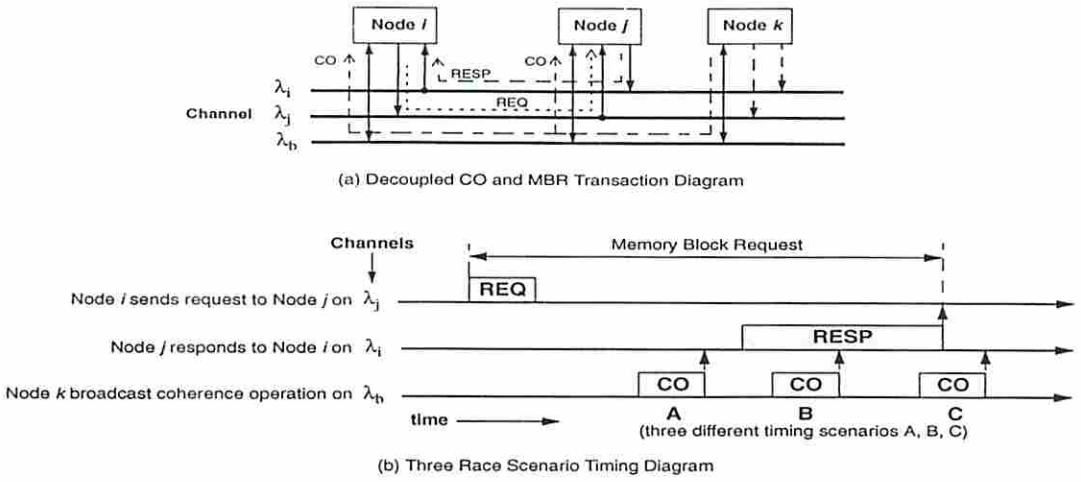

Figure 5: An illustration of possible race scenarios on *DMON*.

That is, the unique owner of a requested memory block is responsible for responding to requests. The ownership of a block is *acquired* by a cache on the event of writing to it (CO) and is re-acquired by memory on *write-back*.

Assumption 5 *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 even for non-critical races and MBRs are not lost during transfer of ownership.

We can now make the following assertion about critical races.

Assertion 1 *A critical race between a CO and a MBR operation on an invalid cache block object results in cache inconsistency.*

Proof 1 The proof follows from considering all case scenarios as given in Figure 5. Figure 5(b) illustrates the timing for the three possible scenarios of a CO racing with a MBR operation, which is possible on (*DMON*) as shown in Figure 5(a). For race scenario **A**, the CO takes effect before the critical period of the MBR operation. From Definition 3, this race is not critical. By Assumption 4 and 5, this type of non-critical race is handled properly and, thus, the RESP comes from the legitimate owner and is up-to-date. In race scenario **C**, the CO loses the race with the MBR operation and, therefore, takes effect on the block which has already been filled in the cache (Assumptions 1 and 2). Here, too, the race is not critical and coherence is maintained.

Application	Number of MBRs	Number of critical races	Critical race (%)
Barnes-Hut	1121444	6027	0.5
MP3D	214596	1088	0.5
LocusRoute	3091541	21836	0.7
Cholesky	4467738	506	0.01

Table 1: Statistics on critical race frequency.

In race scenario **B**, however, the CO wins the race with the RESP (Definition 1, Assumption 2). It therefore completes during the critical period, making this a critical race (Definition 3). By Definition 2 and Assumption 3, the CO fails to take effect on the block being filled while the MBR is in the critical period. Consequently, the received block which now is supposedly stale moves to a valid state once the RESP phase is completed (Assumption 4). By Definition 4, we see that caches are inconsistent as a result of the critical race.

The absence of centralized serialization points in *DMON*'s decoupled architecture is the apparent cause of critical races. Our studies of application traffic on *DMON* indicate that *critical races are vary rare*—on average, less than 0.5% of all MBR operations are involved in critical races (see Table 1). Critical races could be *avoided* by making the RESP phase of MBR operations globally known over the network (i.e., distributed global read/write serialization on every RESP, even though not needed 95.5% of the time), but this produces the opposite desired affect of inefficient channel utilization. Instead, to maintain full concurrency between CO and MBR operations, we allow critical races to occur in *DMON* and propose a cache protocol solution (*SPEED*) which locally *resolves* these infrequent critical race events as they occur.

In Summary: *DMON* supports multiple transmissions on multiple channels over one common optical medium based on passive broadcast and point-to-point wavelength selectivity. A decoupled channel allocation policy is enforced that results in a hybrid interconnect architecture which has the broadcast/synchronization capability of a high-speed bus for coherence operations (i.e., shared writes) and the connectivity of a fully-connected network for non-coherence operations (i.e., read-misses). Although *DMON* is sufficiently scalable and results in balanced channel utilization, its decoupling feature enforces no global serialization which maximizes concurrency but

could lead to potentially hazardous critical race problems. As critical races are shown to be rare occurrences, local detection and resolution by the cache coherence protocol (*SPEED*) proves to be the correct design solution rather than the more restrictive preventive measures. Full concurrency between CO and MBR operations acts to further increase communication efficiency and performance, as confirmed by our results (see Section 4).

3 *SPEED* Cache Coherence

SPEED integrates the virtues of snoopy and directory schemes into one robust protocol designed to exploit the decoupling feature of *DMON*. Instead of serializing coherence and MBR operations targeted for the same memory block at some central/global resource (i.e., at the bus or lock-up directory), *SPEED* allows full concurrency between these operations. This further reduces resource contention and increases communication efficiency. Concurrency of this type, however, can result in races which, if not properly handled, could lead to cache inconsistency and ambiguity in block ownership. *SPEED* has mechanisms which enable nodes to benefit from this added concurrency while autonomously handling potentially harmful races. Hence, our protocol addresses two vital issues: 1) maintaining consistency among cache blocks and main memory in the presence of critical races and 2) maintaining orderly transfer of blocks and block ownership in the presence of critical and non-critical races. In what follows, we describe how *SPEED* addresses these issues and show that it accomplishes the above with comparatively less overhead costs than traditional directory schemes.

3.1 Distinctive Features of *SPEED*

3.1.1 The Pseudo-block

SPEED uses distributed mechanisms in the form of *pseudo-blocks* to detect and resolve critical races as they occur using only locally available information. These mechanisms are used to force a would-be inconsistent copy to remain invalid or to become consistent with other copies before becoming valid (see Figure 6). This can be achieved for a write-invalidate scheme by invalidating the MBR block that loses a critical race after the pending read is performed (Figure 6(b)). This makes the MBR responsive to the CO and effectively serializes the two operations. For a write-update scheme, the updates involved in critical races can be buffered by the node that

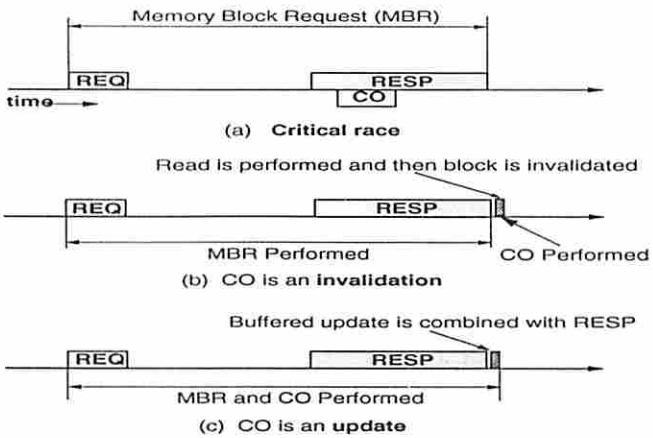

Figure 6: An illustration of critical race resolution.

made the competing MBR and later combined with the received block upon completion of the MBR's RESP phase (Figure 6(c)). In this case, all intervening updates involved in critical races are combined with the received block before the MBR is considered to be performed, again effectively serializing the two operations. Therefore, for both the invalidate and the update versions of *SPEED*, the result is effectively the same as if these operations were serialized on a shared bus. Note that serialization occurs only when absolutely needed—that is, only on critical races.

To cover the critical period of MBR operations, *SPEED* labels a cache block for which a MBR is pending as a *pseudo-block* which, effectively, is a block-fetch-pending state. In accordance with Assumption 3, this state (which is not an invalid state) makes the cache block responsive to any intervening COs. Therefore, the pseudo-block comes into existence during a MBR operation primarily to detect and appropriately represent overlapping operations. Once the pseudo-block is activated for a pending MBR operation, any racing CO which wins the critical race is performed on the pseudo-block which, in turn, dictates the eventual state of the incoming block (as discussed in Section 3.2).

3.1.2 The Owner Directory

SPEED assumes a *write-back* policy. 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 distinctive 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. For instance, caches may acquire block ownership by invalidating copies before

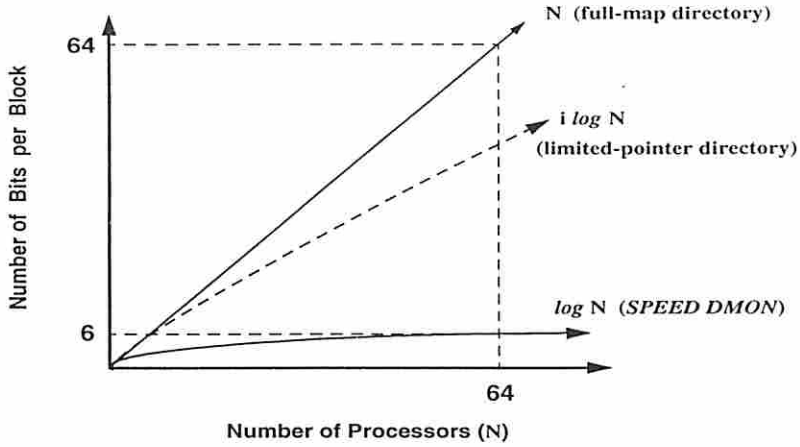

Figure 7: Directory overhead comparison.

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. MBRs are sent first to main memory which, if not the owner, forwards MBRs to the owner as specified by the owner-directory. CO broadcasts are used not only to invalidate or update blocks but also to update the owner-directory. The pseudo-block mechanism guarantees orderly transfer of block ownership in the event of critical races.

SPEED's owner directory has much smaller overhead than other directory schemes, as shown in Figure 7. Only $\log N$ bits per memory block are required to identify an owner, where N is the number of nodes able to cache the block. Full-map directory schemes require N bits per memory block, which certainly is less scalable. *SPEED*'s owner directory overhead matches the minimum hardware overhead required by limited-pointer directory schemes such as LimitLESS [25] which, for the same overhead cost, can store only one pointer before trapping to software overflow handling. Moreover, MBR operations involve only two nodes and do not require synchronization. This greatly simplifies memory controller operations.

3.1.3 Handling Write-Back

Care has to be taken in dealing with write-backs because they involve non-requested transfer of blocks and return of block ownership to main memory. *SPEED* views write-back 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 cache misses in two ways. First, this MBR operation does not have a REQ from main memory. Only a RESP phase exists in which the actual write-back is performed.

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 critical period. Once the pseudo-block is set up, a write-back is performed like a MBR. The timing of ownership transfer is set to the end of a write-back (RESP). An intervening CO results in a critical race. In such a case, the block being written back is discarded immediately. The write-back is considered performed regardless of the outcome of a race, if one occurs. No re-issuance of the write-back is needed as an owned copy must reside in the cache that won the race. The key idea behind this mechanism is that main memory acts like a cache when a write-back is performed.

3.2 The *I-SPEED* Protocol

Here, we describe *I-SPEED*, an invalidate version of *SPEED* (the update version, *U-SPEED*, is beyond the scope of this paper). *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 *exclusive* or *shared*—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 per cache 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 a copy of the block has been forwarded. Ownership remains unchanged. Note that the cache to which the copy was 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 created 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 created due to a write-miss is *pseudo-owned*. The *pseudo-owned* state is used to cover the period over which the owner does not have the actual block.

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

The first four states are based on traditional concepts put forth in [26, 24]. We introduce the notion 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 invalidation of the block being transferred after the pending read is performed. The *pseudo-owned* state allows two different handling of write-misses as described in the following section.

3.2.1 Cache Procedures

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

- Read-hit: Read. No state transition is involved.

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

- Write-miss: A write-miss can be handled in two ways. For both schemes, broadcasting is used only once; however, the first scheme trades off simplicity for a reduced number of channel accesses and latency.

    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 write-miss is performed as a sequence of a read-miss followed by a write-hit to the block (see above procedures).

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

I-SPEED handles write-misses in two ways. In one scheme, which we refer to as *I-SPEED Combined*, a MBR combined with an invalidation (REQ+INV) is broadcasted on a write-miss. This is similar to most write-invalidate protocols [26]. In this scheme, a pseudo-block (with *pseudo-owned* state) is activated at the end of the REQ+INV phase and is used to detect and serialize locally any overlapping MBRs (both from read and write misses). Overlapping MBRs are not responded to until the pending write is performed, effectively serializing overlapping MBRs and avoiding critical races. This scheme offers two advantages. First, the number of channel accesses for a write-miss operation can be reduced at least by one as REQ is combined with INV. Second, the latency of sending requests can be reduced as requests are broadcasted to the owner without having to refer to the owner-directory. However, a disadvantage of this scheme is that it requires buffering of overlapping MBRs.

The second scheme, which we refer to as *I-SPEED Combo*, performs the write-miss as a sequence of a read-miss (MBR) followed by a write-hit (INV). Few, write-invalidate protocols implement this due to network constraints. In this scheme, write-misses, per se, do not exist.

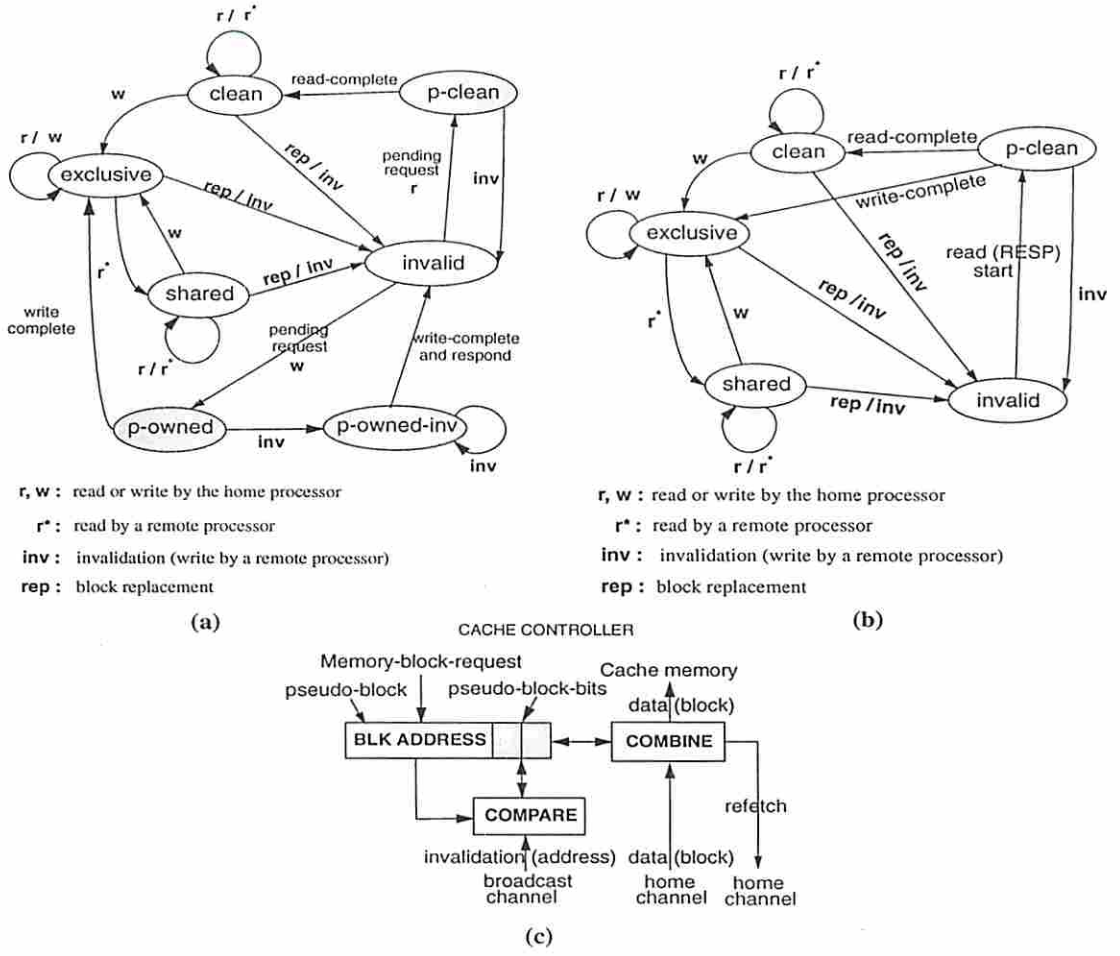

Figure 8: (a) State transition diagram of *I-SPEED Combined* and (b) state transition diagram of *I-SPEED Combo* and (c) mechanism for pseudo block in cache controller.

The pseudo-block (with *pseudo-clean* state) is used to cover critical races for MBRs of both regular read-misses and read-misses resulting from write-misses. The advantage of this scheme is its simple operation. A disadvantage, however, is that at least one more channel access is required for the separate MBR operation. Also, the latency of sending a request can be longer due to owner-directory access.

3.2.2 Cache State Transitions

In what follows, we describe the state transitions for the *I-SPEED Combo* variant of our protocol. Figure 8(a,b) illustrates the state transition diagram of cache blocks and pseudo-blocks for *I-SPEED Combined* and *Combo*. Figure 8(c) shows a conceptual implementation of the pseudo-block mechanism for a node when a single outstanding MBR is allowed. Race problems for multiple outstanding requests can be handled by increasing the number of pseudo-blocks allowed

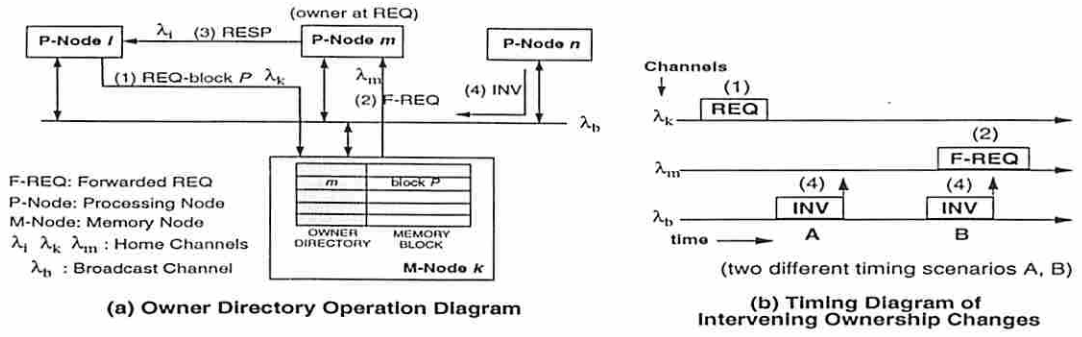

Figure 9: (a) Diagram of owner-directory operation for a MBR and (b) non-critical race scenarios.

per cache.

Although the *pseudo-clean* state is used for both read and write misses with *I-SPEED Combo*, critical races for MBRs of write-misses require different handling from that of read-misses because write-misses involve transfer of ownership. On a read-miss, a pseudo-block is activated in the *pseudo-clean* state; an intervening invalidation resets the pseudo-block to *invalid*. In such a case, the pending read which issued the request for the MBR is allowed to use the block before it is invalidated. In the case of a write-miss, a pseudo-block is again created and set to the *pseudo-clean* state to cover the read-miss part of the pending write-miss. However, the read-miss (and subsequent write-hit) is *not* allowed to complete in the event of a critical race. Doing otherwise would result in the incorrect transference of ownership of this stale block. Hence, the termination and re-issuance of MBRs involved in critical races is required for write-misses.

3.2.3 *SPEED* Applied to *DMON*

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 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 owner-directory. Processor-Node m responds to Processor-Node i. All the operations above are performed on the *home* channel of each node. Because REQs are sent on *home* channels, invalidations overlapping with the forwarding of REQs can lead to non-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. In this event, 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 back to the requesting node. Our simulation results indicate that this type of non-critical race is very rare. Therefore, *SPEED* can adopt either scheme with little difference in overall performance.

Table 2 summarizes the critical and non-critical races for all combinations of cache and memory operations and lists *SPEED* solutions. The first two columns specify pending operations and their phases. The next two columns list those racing operations and their phases that can transpire simultaneously on other *DMON* channels. Finally, the last two columns list the potential problems resulting from these racing operations and our proposed solutions.

3.2.4 Handling Exceptions

We mention below an exceptional case that could arise in the proposed critical and non-critical race solutions. Resolution involves re-issuing MBRs and forwarding or negatively acknowledging REQs. This could result in a form of livelock where the solution or combination of solutions is pathologically repeated. In particular, the critical race or the non-critical race involving the change of ownership could be repeated continuously for MBRs from the same miss on a heavily shared block, causing undue delay in one or more nodes. Although our simulation results indicate that such events are extremely rare[2], their possibility should not be ignored. One solution is to have some sort of *count* mechanism to limit the number of repetitions before one of the following exception handling procedures is invoked: back-out of repeated critical races by deferring invalidation or broadcast the repetitious MBR so as to not allow invalidations to race with the MBR.

4 Performance

We evaluate the effectiveness of *SPEED DMON* by comparing its performance to that of invalidation-based snoopy and directory cache coherence schemes on a multi-channel optical network (*MON*). One of the most important factors that affects the performance of cached DSMs is the *network*

[2]In fact, no such livelock events have been detected in our simulation.

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 2: Summary of critical and non-critical races and *SPEED* solutions.

latency of MBR operations and COs. For the optical interconnects of our interest, three major components make up network latency; channel *arbitration* latency, *transmission* latency, and latency due to channel *contention.*

Channel arbitration latency depends very much on the arbitration scheme used. Under an arbitration scheme, it is the sensitivity of a cache protocol's network latency to channel arbitration that is of importance (i.e., which cache coherence scheme incurs more arbitration latency than others). Transmission latency of a message for each channel hop is determined by channel bandwidth and message size. Therefore, for a given channel bandwidth and message size, the aggregate transmission latency depends solely on the number of channel hops the message take to complete a transaction. With a limited number of channels and bandwidth, two factors determine channel contention; *traffic*, and *channel allocation.* For a given bandwidth and number of channels, channel contention is most affected by efficiency in channel allocation for a given traffic load.

We are also interested in determining whether *SPEED DMON* has potential to scale better than other schemes, particularly directory-based schemes. In the following section, we describe the evaluation methodology.

4.1 Evaluation Methodology

Our cache and network simulator is built on top of MINT [27], a program-driven multiprocessor simulation platform. Systems with an arbitrary number of nodes optically interconnected by *DMON* and *MON* can be simulated. The two main modules of the simulator are:

Node Module Architecture: A node consists of a processor, a single-level cache, a main memory segment which includes a distributed shared memory segment and private memory, a local bus, and a network interface. The cache system for shared data is a direct-mapped write-back cache (4 KB with $32byte$ blocks). Transactions between caches and main memory are either local (within the same node requiring 2 pclocks on the local bus) or remote (through the network). Prefetching is not assumed in this simulation. Thus, at most one outstanding MBR per each processor is allowed at a given time. Table 3 summarizes the size of messages that caches and main memory generate.

Network Module Architecture: *DMON* based on tunable transmitters is used for this evaluation. Although the same network resources are assumed for *MON* (see Section 2 *DMON*

Operation Type	Message Size
Request (REQ), Invalidation (INV), Acknowledgment (ACK)	8 *bytes*
Block response (RESP), Block replacement (RESP)	36 *bytes*
Local transfer (all transaction)	Uniform 2 pclocks transaction time

Table 3: Network message sizes.

Application	Input size
Barnes-Hut	8k bodies, 4 iterations
MP3D	10k particles, 5 iterations
LocusRoute	3817 wires, 20 routing channels (Primary2.grin)
Cholesky	3948 by 3948 floats, 56934 non-zeros (BCCSSTK15)

Table 4: Application Programs.

description), the way in which they are used differ. For the snoopy protocol (*Snoopy*), the tunable transmitter resource is used as a second *broadcast* channel which, in effect, doubles the broadcast bandwidth. For the directory protocol (*Dir*), the broadcast channel is disabled[3]. The tuning delay of tunable transmitters are fixed at 10 ns [5]. We will discuss how a longer tuning delay may affect the network latency of each protocol (*SPEED* and *Dir*) in the following section. Channels are arbitrated by a channel *reservation* scheme similar to that proposed in [12]. An arbitration slot in which arbitration or reservation for channel accesses can occur is allocated to each node in a round-robin fashion. To minimize arbitration overhead, the arbitration slot size is set to one processor cycle. Our network simulator accurately models channel contention. Reservations are made for channel accesses (*home* and *broadcast*) that result in contention.

Throughout this evaluation, the processor clock (pclock) cycle is used as the base time unit for all measurements. *Snoopy* and *Dir* used in this simulation are characterized as follows:

Cache Coherence Protocols: As in *SPEED*, an *exclusive* cache state is used for both protocols to minimize the number of network accesses on writes. *Snoopy* also assumes *split-transactions* to minimize the *broadcast* channel hold-up for MBRs. We use a full-map directory for *Dir* to avoid modeling software overhead dealing with directory overflow. Directory contention is modeled by locking busy entries. In the event of directory contention, accesses are negatively acknowledged to be retried.

Benchmarks: Four parallel applications taken from the Stanford SPLASH suite [28] are used

[3]This has little effect on the overall bandwidth of the network because a fixed-frequency channel can only be used as an additional *home* channel.

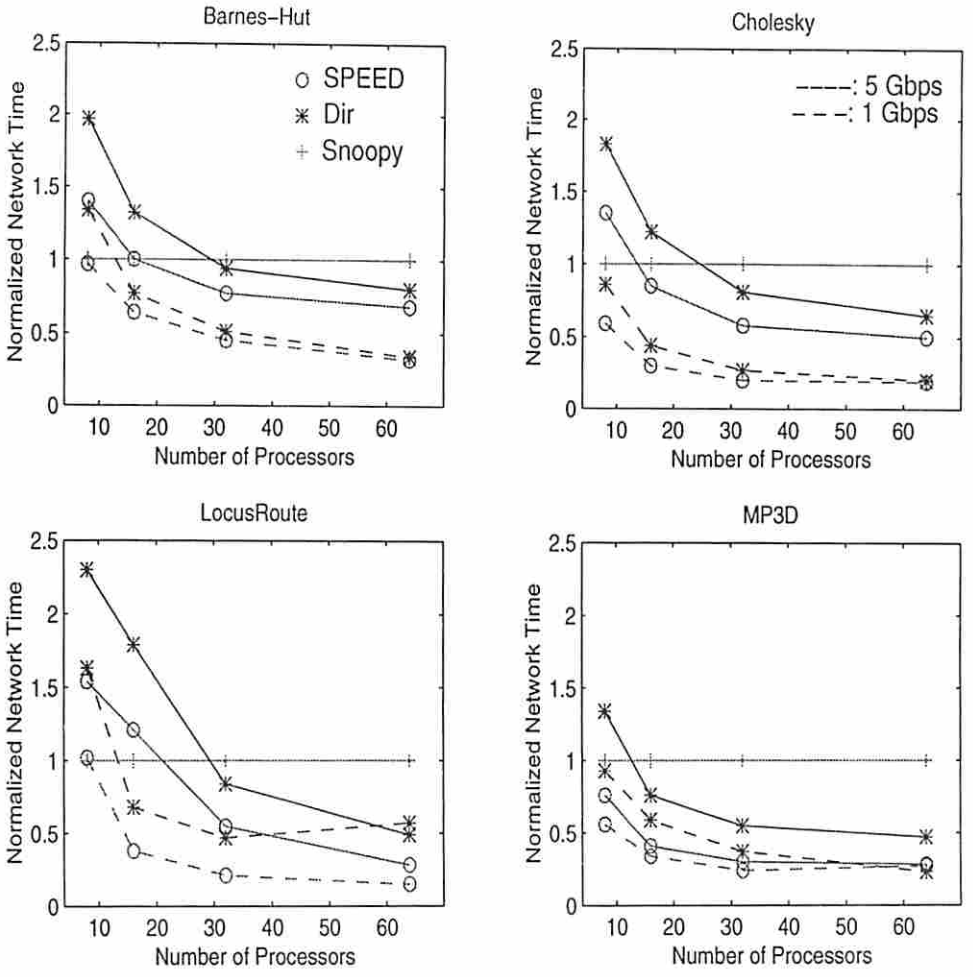

Figure 10: Relative normalized network latency among Snoopy, Directory, and *SPEED*.

(Table 4) in this evaluation.

4.2 Experimental Results

Other than the results presented for intuitive performance prediction presented in Section 2.1, we use three further metrics to evaluate the performance of *SPEED DMON*: *network latency reduction*, *relative channel efficiency*, and *sensitivity to channel latency*.

Network Latency Reduction: Figure 10 shows the network latency of *SPEED* and *Dir* normalized to that of *Snoopy* for 1 and 5 Gbps channel bandwidth. Figure 11 shows normalized network latency broken down into 6 components, for a 64 processor system. Each of these components represents how much time on average each processor spends on channel *arbitration*, channel *contention*, and *transmission* for both MBRs and INVs to complete the execution of

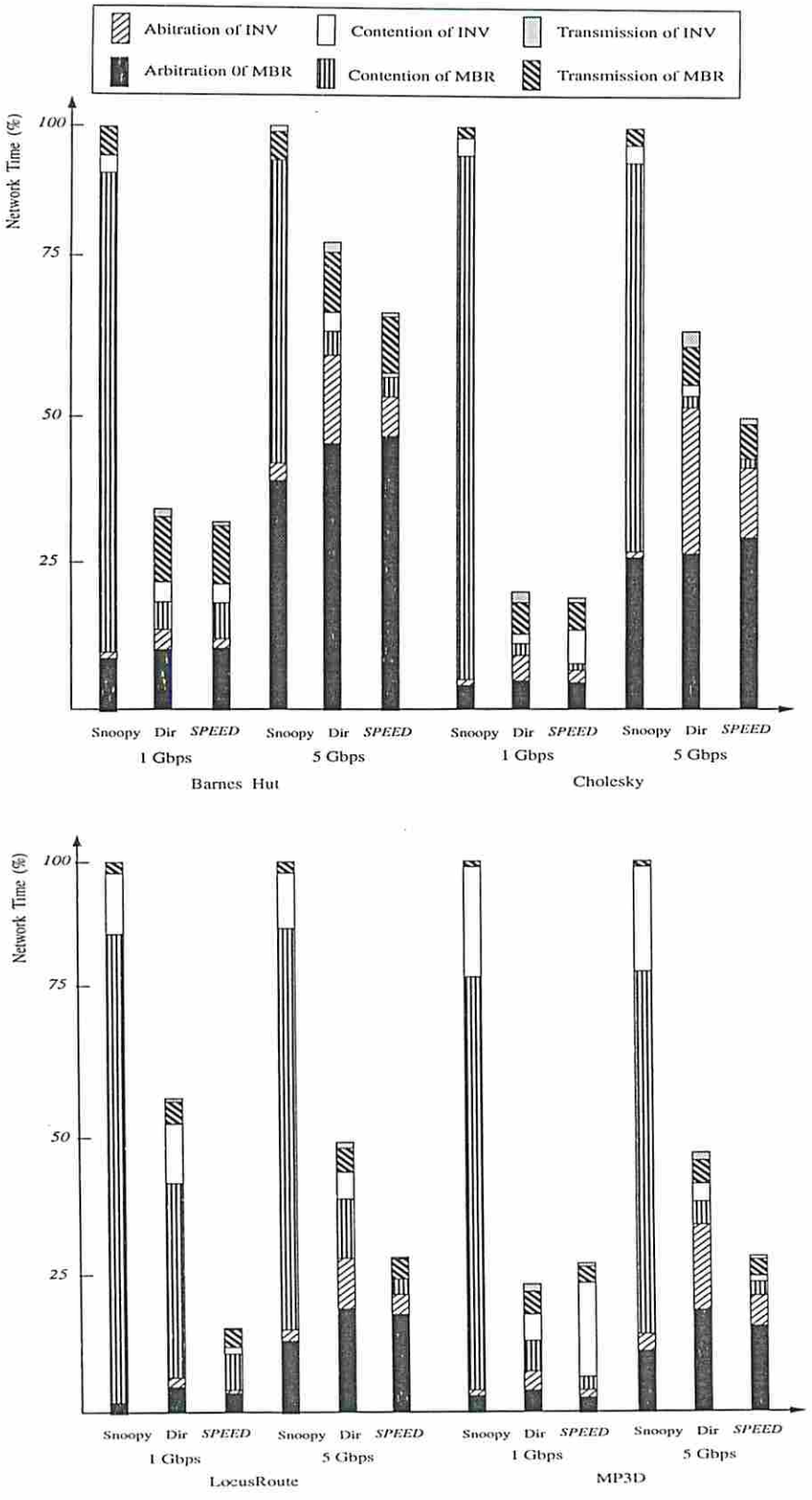

Figure 11: Network latency break-ups among *SPEED*, *Snoopy* and *Dir* for 64 processors.

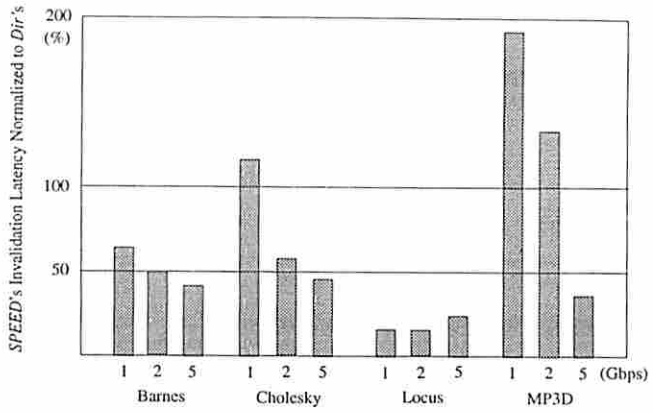

Figure 12: *SPEED*'s invalidation latency normalized to *Dir*'s: decoupled invalidation broadcasting versus point-to-point invalidation.

an application. *SPEED* clearly reduces network latency up to 85% as compared to *Snoopy* for systems with more than $\approx$ 20 processors. Higher network latency reduction is expected particularly in the case of high and temporally concentrated miss traffic which would saturate the bus (e.g., LocusRoute and Mp3d). The performance advantage is lower for low and distributed miss traffic (e.g., 68% reduction for Barnes-Hut). *SPEED*'s advantage over *Snoopy* is clearly due to the contention for MBRs, which increases with increasing number of processors. On the other hand, *Snoopy* does have two benefits. One is that it is least sensitive to channel arbitration owing to its single-hop operations (i.e., each channel hop involves channel arbitration). The other is that the single-hop operation of snooping pays off for small numbers of processors (less than $\approx$ 10) if sufficient broadcast bandwidth is provided (e.g., 5 Gbps).

SPEED also achieves lower network latency than *Dir* in most cases and the latency gap somewhat widens as the channel bandwidth increases. We observe that the decoupling contributes to this in two different ways. Firstly, the decoupled broadcasting of INV cost less in most cases than sending invalidations only to those nodes caching the block. Figure 12 shows *SPEED*'s aggregate INV network latency normalized to *Dir*'s. In the figure, decoupled INVs of *SPEED* outperforms the pont-to-point INVs of *Dir* if a sufficient *broadcast* channel bandwidth is provided ($\approx$ 3 Gbps). With 5 Gbps channels, the *broadcast* channel contention for INVs is negligible (Figure 11), which indicates that more processors can be accommodated for the available bandwidth. Furthermore, *Dir* is inherently more sensitive to arbitration because INVs require at least 1 more channel hop for directory accesses and could require a total of 4 hops. Secondly, by using the dedicated network for MBRs, the latency of MBRs is also reduced, but only marginally because of the relatively small bandwidth requirements (Section 2.1) of INVs and the small number of

Application	*Home* channels for *Dir*	*Home* channels for *SPEED*	Efficiency
Barnes-Hut	64	10	6.4
MP3D	64	16	4
LocusRoute	64	8	8
Cholesky	64	16	4

Table 5: Relative channel efficiency for the 64 processor system.

sharers of each block (≈ 2). LocusRoute appears to be an exception (Figure 11). With *Dir*, LocusRoute shows much more contention for MBRs than other applications because INVs interfere with MBRs more (that is, MBRs and INVs are temporally concentrated). As the results show, the decoupling works best in such cases.

Relative Channel Efficiency: We measure relative channel efficiency between *SPEED* and *Dir* by the ratio of the number of channels that the two protocols require in order to achieve comparable aggregate channel contention for MBRs and INVs. In Table 5, *SPEED* requires less than 25% (16 channels) of the number of channels that *Dir* requires (64 channels) to achieve a comparable contention latency. This means that, by the decoupling and by adding one more channel for broadcasting, the number of *home* channels that *SPEED* requires to achieve the performance comparable to *Dir*'s is reduced drastically (by more than 75%). This result has a significant implication in terms of scalability. That is, with *SPEED*, more processors can share the same *home* channel with significantly less effects on the overall network latency. Therefore, with a given number of available channels, *SPEED DMON* is more scalable.

Channel Latency Sensitivity: Channel latency sensitivity represents how sensitive the network latency is to the variation of channel latency. We are particularly interested in measuring protocol's sensitivity to the *home* channel latency because the transmitter tuning delay can change *home* channel latency significantly in systems using tunable transmitters. *Home* channel latency sensitivity is measured in terms of percentage increase in the *home* channel contention when the channel bandwidth is reduced (in this case, from 5 Gbps to 1 Gbps). Figure 13 shows that *SPEED* is less sensitive to channel latency than *Dir* except for Barnes-Hut. This result indicates that the effect of transmitter tuning delay should have less effect on overall network latency with *SPEED DMON* in most cases.

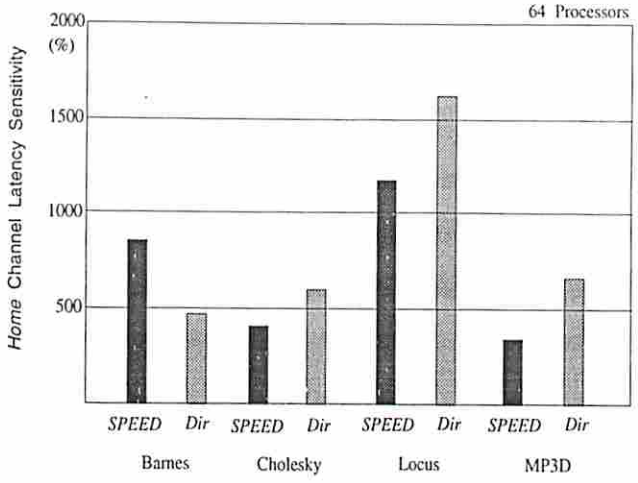

Figure 13: Protocol's sensitivity to the *home* channel latency (*SPEED* and *Dir*).

5 Related Work

The efficient use of multiple WDM-based optical channels for cached DSMs has been studied by other researchers. Ghose et al. [29, 14], 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 globally serializes operations at main memory. Dowd et al. [12], studied the interaction between different channel access control and cache coherence protocols for DSMs. 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. Nowatzyk et al. [15] proposed a time-division multiplexing-based fully-connected optical interconnect which allows the design of cache systems for DSMs to take advantage of the scalable broadcasting of optics. Their analysis generally agrees that optics provide unique opportunities to simplify cache coherence protocols and synchronizations in DSMs. Hence, our work seems to stand alone in its approach to developing a hybrid cache coherency protocol for multi-channel interconnect architectures based on optics.

6 Conclusion

SPEED is an integrated cache coherence protocol designed to exploit high bandwidth point-to-point and broadcast features provided by our proposed decoupled multi-channel optical network (*DMON*). *SPEED* uses directory-assist for memory block requests (e.g., read-misses) to eliminate unnecessary broadcasts and uses snoopy-assist for coherence operations (e.g., writes) to reduce directory overhead and synchronization complexities. Given typical communication behavior of shared memory multiprocessors, 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 DMON* 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 *DMON*'s 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 as compared 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 research is needed in the verification of our *SPEED* protocol. An update-based version of *SPEED* (*U-SPEED*) is being designed that may make even better use of the high bandwidth potential of optics. Finally, guided-wave and free-space implementations of *DMON* 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*, 8:984–964, August 1990.

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

[3] K. Kato et al. "Packaging of large-scale integrated-optic N*N star couplers." *IEEE Photonics Technology Letters*, 5(6):348–351, 1993.

[4] K. Rastani, C. Lin, and J. S. Patel. "Multiple Bus Architectures." *Applied Optics*, 31(16):3046–3050, June 1992.

[5] B. S. Glance, J. M. Wiesenfeld, U. Koren, and R. W. Wilson. "New Advances on Optical Components Needed for FDM Optical Networks." *IEEE Photonics Technology Letters*, 5(10):1222–1224, October 1993.

[6] D. A. B. Miller. "Optics for low-energy communication inside digitial processors: quantum detectors, sources, and modulators as efficient impedance converters." *Optics Letters*, 14(2):146–148, January 1989.

[7] Susan J. Eggers and Randy H. Katz. "A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation." In *Proceedings of the 15th International Symposium on Computer Architecture*, pages 372–383. IEEE Computer Society Press, 1988.

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

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

[11] Anoop Gupta, Wolf-Dietrich Weber, and Todd Mowry. "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes." Technical Report CSL-TR-90-417, Stanford University, March 1990.

[12] 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.

[13] Timothy Mark Pinkston, Seelan Kumarasamy, and Charles Kuznia. "Hybrid CMOS/SEED Channel Flow Controller Chip." In *Submitted to the 1996 International Topical Meeting on Optical Computing.*

[14] 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, August 1994.

[15] Andreas G. Nowatzyk and Paul R. Prucnal. "Are Crossbars Really Dead? The Case for Optical Multiprocessor Interconnect Systems." In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 106–115, June 1995.

[16] 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.

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

[18] 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.

[19] 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.

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

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

[22] 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.

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

[24] 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.

[25] 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.

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

[27] Jack E. Veenstra and Robert J. Fowler. "MINT: A Front End for Efficient Simulation for Shared-Memory Multiprocessors." In *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 201–207, January 1994.

[28] J-P. Singh, W-D. Weber, and A. Gupta. "SPLASH: Stanford parallel applications for shared-memory." *Computer Architecture News*, 20(1):5–44, March 1992.

[29] 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.