To appear in Proceedings of International Conference on Computer Design, Austin, Texas, October 1996.

Global Bus Design of a Bus-Based COMA Multiprocessor DICE

Gyungho Lee, Bland Quattlebaum,
Sangyeun Cho†, and Larry Kinney

Dept. of Electrical Engineering
†Dept. of Computer Science
University of Minnesota
Minneapolis, MN 55455
E-mail: ghlee@ee.umn.edu

Abstract

DICE is a shared-bus multiprocessor based on a distributed shared-memory architecture, known as Cache-Only Memory Architecture (COMA). Unlike previous COMA proposals for large-scale multiprocessing, DICE utilizes the COMA to effectively decrease the gap between modern high-performance microprocessors and the bus. As microprocessors become faster and demand more bandwidth, the already limited scalability of a shared bus decreases even further. DICE tries to optimize the COMA for a shared-bus medium, in particular to reduce detrimental effects of the cache coherence and the "last memory block" problem on replacement. In this paper, we present a global bus design for a bus-based COMA multiprocessor using the IEEE Futurebus+ standard backplane bus and the Texas Instruments chip-set. Our design demonstrates that necessary bus transactions for DICE can be done efficiently with existing standard bus signals. Considering the benefits of the COMA and the little design complexity it adds to the conventional shared-bus multiprocessor design, a bus-based COMA multiprocessor such as DICE can become a viable candidate for future shared-bus multiprocessor designs.

1 Introduction

Shared-bus SMPs (Symmetric Multi-Processors) such as the Sequent Symmetry [14] or the SGI Challenge [3] represent the mainstream of accepted and commercially viable computer systems. However, as microprocessors become faster and demand more bandwidth, the already limited scalability of the shared bus decreases even further, and the ill-effect of a cache miss penalty becomes even worse. Even with clustering of having several processors per a processor board, the effective machine size for shared-bus multiprocessors is fairly limited. Further, a cache miss can cost up to a few hundred processor cycles for recent high-performance microprocessors. To bridge the gap between high-performance microprocessors and a backplane bus, it is important to reduce global bus traffic and to increase local memory utilization, together with efforts to develop a high-speed wide data-path backplane bus.

The DICE (Direct Interconnection of Computing Elements) project at the University of Minnesota utilizes the *Cache-Only Memory Architecture* (COMA) to bridge the gap. The COMA improves the utilization of local memory by decoupling the address of a datum from its physical location, allowing the data to move dynamically beyond the level provided by traditional caches. This decoupling is achieved by treating the memory local to each node, called *attraction memory* (AM), as a cache to the shared address space without providing traditional physical main memory [5].

Unlike the previous examples of scalable COMA machines, including the DDM of the Swedish Institute of Computer Science [5] and the KSR-1 of the Kendall Square Research [23], DICE focuses on the efficient realization of the COMA as a shared-bus SMP with little provision for scalability for larger-scale multiprocessing. While we expect many problems associated with scalable COMA machines to become less serious with a shared-bus medium, shared-bus multiprocessors benefit from the COMA in three ways: (i) less bus contention due to lower global traffic; (ii) shorter average memory latency due to higher local memory utilization; and (iii) more processors in the machine due to less bandwidth requirement on the bus.

This paper presents a global bus design of DICE. The

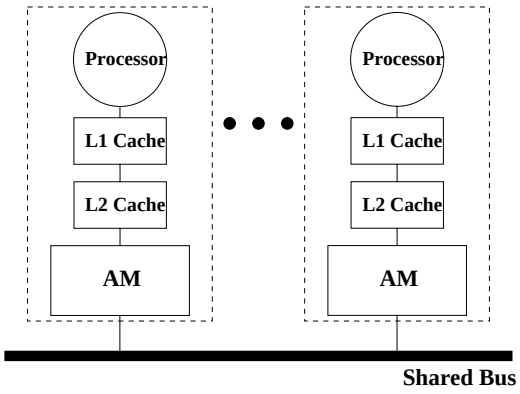

Figure 1: A bus-based COMA multiprocessor

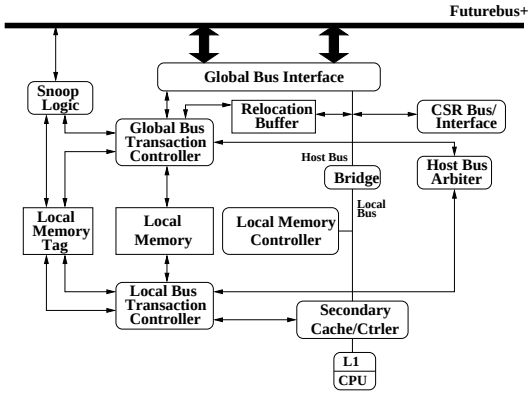

Figure 2: Block diagram of a DICE node

main contribution of this paper is in demonstrating the feasibility of an efficient implementation of a bus-based COMA multiprocessor. Especially, we focus on how the implementation handles the coherence enforcement and the replacement problem, which can cause significant overheads in scalable COMA machines [7, 9]. Our design employs the IEEE Futurebus+ standard backplane bus [22] and the Texas Instruments chip-set [21].

The rest of this paper is organized in the following manner. Section 2 gives a brief background necessary for our discussions. Section 3 describes the coherence and replacement protocol of the DICE multiprocessor. A global bus implementation is given in Section 4, and Section 5 will summarize the paper.

2 Background

2.1 Why bus-based COMA?

Shared bus design has been popular in small-scale commercial SMPs. A commercial SMP typically runs a single instance of an operating system with a shared real address main memory, and supports hardware cache coherence control. Among recent machines are the SGI Challenge [3] and the Sun Microsystems Ultra X000 Servers [24]. Although the shared-bus SMP is a widely accepted architecture, its scalability is severely hurt due to the limited bandwidth of the bus. As microprocessors become faster and demand more bandwidth, the shared bus becomes an even more serious bottleneck in such systems. If the last ten-year history is any indication for future, then one should expect that the bottleneck will become even worse. For example as noted in [6], with 16 processors, a block size of 64 Bytes, and a 64-KB data cache, the total bandwidth demand for some parallel benchmark programs ranges from almost 500 MB/sec (for Barnes in SPLASH-2 [20]) to over 9400 MB/sec (for Ocean), assuming a processor that issues a data reference every 5 ns. In comparison, the Gigaplane bus of the Ultra X000 Servers, one of the highest bandwidth bus systems, provides 2500 MB of bandwidth [24].

Handling the problem of the shared-bus bottleneck can be done in three complementary approaches. Firstly, a faster and wider bus needs to be developed. This can be achieved by developing low voltage-swing bus transceivers, high density packaging, effective grounding to reduce noise interference, and more effective line termination. Secondly, smart bus protocols such as more aggressive pipelining are needed. Thirdly, memory requests should be serviced locally. This end can be met by having larger caches (of multi-level structure) or large shared caches together with "clustering" [16]. Taking this to an extreme, a DSM (Distributed Shared Memory) architecture, especially the COMA, becomes attractive.

The COMA improves the utilization of local memory by decoupling the address of a datum from its physical location, allowing the data to move dynamically beyond the level provided by traditional caches. With dynamic replication and migration of data through the AMs, a COMA machine seems to be able to provide higher utilization of local memory than is otherwise possible, which may result in low average memory access latency and low network traffic. As the processor technology is progressing much faster than the bus or interconnection network technology, this potential reduction in latency and bandwidth requirement can be a crucial advantage.

2.2 Bus-based COMA multiprocessors

Figure 1 shows a high-level structure of a bus-based COMA multiprocessor. A processor node (dashed box) is composed of a high-performance microprocessor, two levels of cache memory, and the local memory managed as the AM. The local memory tag, which includes 'state' information and uses fast SRAMs, is duplicated so that local tag access and global bus snooping will not conflict too often

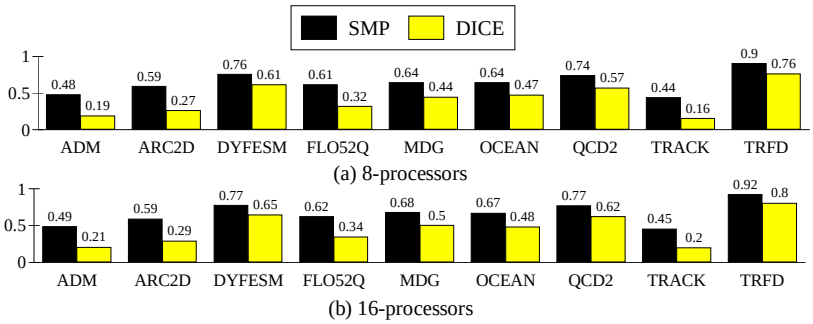

Figure 3: Global bus utilization (memory pressure = 60%)

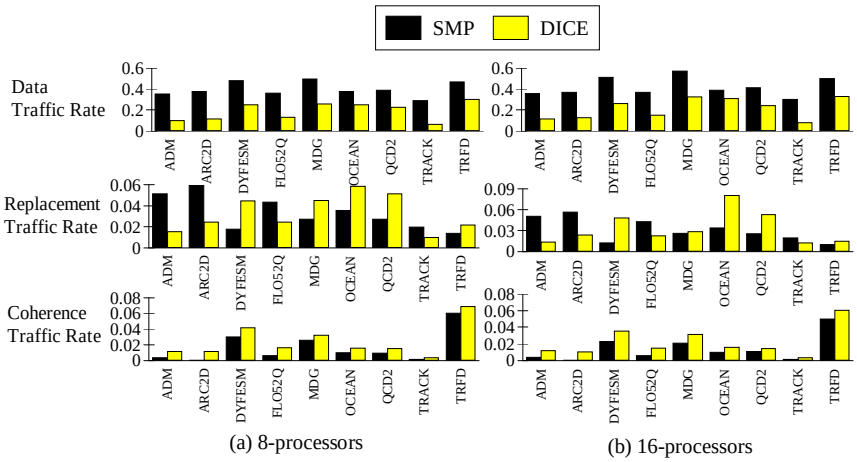

Figure 4: Bus traffic rate per reference (memory pressure = 60%)

at the tag. The inclusion property [1] is maintained in the memory hierarchy. Figure 2 gives a block diagram for a DICE node.

As in a traditional shared-bus machine, every node snoops all global bus traffic. In dealing with large AM, it can be challenging to have a snoop control logic that can keep up with a modern backplane bus with a high clock frequency, especially if the memory access model is based on the sequential consistency [10]. For example, with the SGI POWERpath-2 [3], each bus transaction takes five clock cycles of the 47.6 MHz clock, and the snooper has about 35 ns (less than two cycles) to search the state and tag for its AM and then update the state if necessary. With the fast SRAMs currently available, the snooper has little difficulty in keeping the AMs of the COMA coherent. However, if the snooper cannot keep up with the fast clock of the backplane bus, one can adopt relaxed memory models such as the release consistency [4] in order to perform the snooping asynchronously and delay the coherence actions. Note that recent high-performance microprocessors support the relaxed memory models [6].

A major overhead involved in the COMA is additional extra memory. One source of this need for extra memory is the state and tag memory for the AM. While it is not significant in terms of the amount of space, the state and tag memory can be a significant overhead in terms of cost. Extra memory is also needed for the unallocated space, which has been reported to be essential for good performance of the COMA. Since the unallocated space is necessary mostly for shared variables, its amount can be kept reasonably small [7, 8, 9], especially when the set-associativity for the AM is four or higher.

2.3 Potential performance

To gauge the potential performance advantage of a bus-based COMA multiprocessor over the traditional bus-based SMPs, we have simulated a “scaled-down” DICE machine labeled *DICE* in Figure 3 and a traditional shared-bus multiprocessor modeled after the SGI Challenge [3] labeled *SMP*. The effects of contention at the processor cache, at the local memory, and at the shared bus are reflected in our simulation results. A detailed description of our simulation environment and results are found in [13].

Figure 3 and 4 show the bus utilization and traffic rate per reference for the studied architectures respectively. For

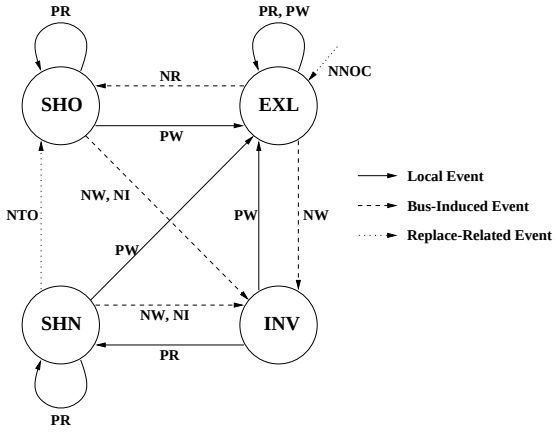

Figure 5: DICE write-invalidate coherence protocol (PR: Processor Read, PW: Processor Write, NR: Network Read, NW: Network Write, NI: Network Invalidation, NTO: Network Transfer of Ownership, NNOC: Network No Other Copy)

the nine programs from the Perfect Club Benchmark [2], our simulation results show significant bus traffic reduction. *DICE* generated slightly more traffic for replacement and coherence for some programs. The results are consistent with the results of our previous study [12]. A recent study [11] on a bus-based COMA multiprocessor reports a similarly significant reduction in bus traffic: a traffic reduction of up to 70%, with an average of 46%, for the six SPLASH benchmark [19] programs.

3 Coherence and Replacement

In this section, we outline the coherence and replacement protocol for the DICE multiprocessor. More details of our coherence and replacement protocol are found in [13]. We discuss major aspects of the protocol, which is different from the one for traditional SMPs.

Figure 5 shows the four-state write-invalidate coherence protocol for DICE. An AM block can be in any one of the four states: Invalid (INV), Shared Non-owner (SHN), Shared Owner (SHO), and Exclusive (EXL). The SHN state is a non-owner state and guarantees that the block in this state is not the only copy in the system. The SHO state is an owner state and carries an ambiguity – there may or may not be other copies. The EXL state guarantees that the block is the only copy in the system, and ownership is implicit. The SHO and EXL states indicate the responsibility of supplying data when a read or write request for the block is seen on the bus.

Ownership removes the ambiguity in responding to bus transactions (e.g., on an AM miss) and reduces the traffic related to memory block replacement, which poses a unique problem in COMA multiprocessors. A falling-off block due to replacement, if it has ownership, needs to transfer its ownership to a shared copy if any, or relocate to a remote node if it is the "last copy" of the memory block. Although the cache-like local memory can be backed up by system disk(s) on replacement, its tremendous overhead prohibits such operations.

On a reference miss, a (victim) block in the set to which the reference maps has to be selected to receive the incoming data. Unlike LRU or random selection in traditional caches, the states of the blocks are used to choose the victim prioritized in the following order: INV, SHN, SHO, EXL. The INV and SHN states do not incur the relocation process. Figure 6 shows the results of this priority-based selection for several cases assuming 4-way set-associative AMs. Victim candidates after the priority-based selection are marked with a darker block, and the victim is selected randomly should there be more than one candidate. If the selected victim is in the SHO or EXL state, it needs to be relocated. A priority scheme is used in choosing which node to accommodate the block to be relocated. Figure 7 briefly demonstrates our priority scheme. A node with a shared copy of the replaced block is given the highest priority. It is clear that this case is possible only when a block in the SHO state is replaced. Ownership transfer without an AM update suffices in this case. The second priority is given to the node with a block in the INV state and no shared copy of the replaced block. The data will be stored in the block frame, and the resulting state is EXL regardless of the state that the original replaced block had. Next priority is given to a node with a SHN block which is not identical to the replaced block. The lowest priority is given to a node with blocks all having ownership. To avoid the chain of relocation, a processor node which originates relocation can acquire ownership of the incoming data, so that the block to be relocated may not go down to the lowest priority case [13]. It may seem that relocation to the node with a block in the INV state is preferable to the node having a shared copy of the replaced block. However, our scheme favors a node with a shared copy because (i) relocation incurs ownership transfer only, and (ii) better performance can be achieved from the efficient use of memory space [7].

4 A Global Bus Design

We present in this section a global bus design for DICE based on our previous discussions. A complete description of this section can be found in [17] and [18]. Our design uses the IEEE Futurebus+ standard bus. The implementation presented here is one of many possible implementations. Although the design described in [17, 18] uses

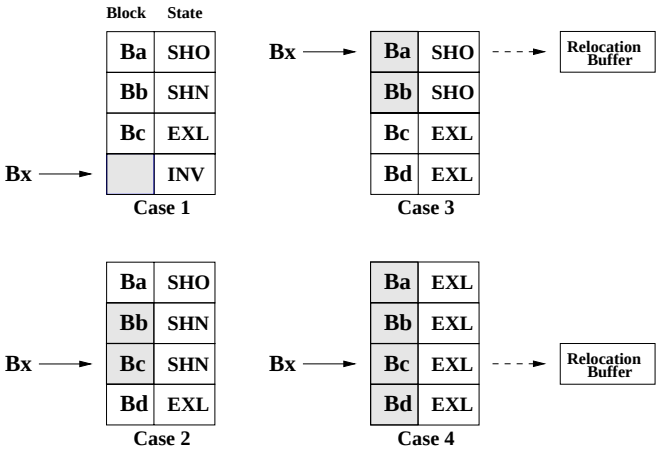

Figure 6: Victim block selection (B_X: incoming block to satisfy a miss)

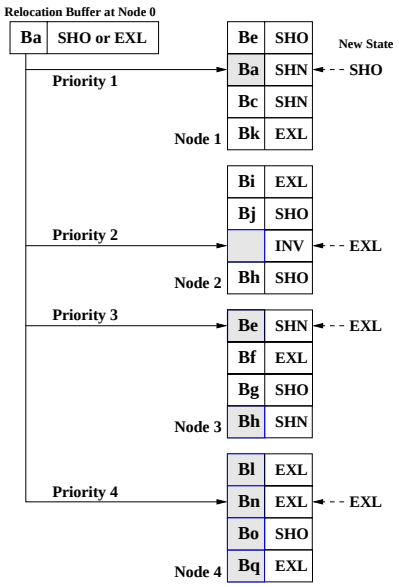

Figure 7: Selecting a remote AM for relocation

a *write-update* policy, our discussion is limited to the one with a *write-invalidate* policy.

4.1 FB+ background

We chose the Futurebus+ (FB+) [22] for our global bus implementation. In the discussions which follow, the implementation uses the B-profile specification detailed in IEEE 896.2 for a couple of reasons. The profile B supports a distributed arbitration protocol, which is desirable not only to remove the poor system scaling associated with a central arbitration but also for the replacement and relocation algorithm. Moreover, several companies including Mupac, Schroff, and Texas Instruments (TI) [21], offer profile B compliant chip-sets, backplanes, and Eurocard enclosures. This greatly simplifies the bus interface design by providing a proven implementation of the profile.

Table 1 shows the transaction mapping between those proposed to support the DICE multiprocessor and those provided by the FB+. To enhance the capabilities of the basic bus transactions, the IEEE 896.1 specification provides eight user-defined signal lines, `TAG[7:0]`. In addition, two modes of data transfer are provided on the bus, namely *packet mode* and *compelled mode*. The first allows up to a 64-contiguous-byte transfer using only the address of the first word. The compelled mode on the other hand requires a handshake for each data transfer. The *Read/Write Unlocked* transactions may be used in the packet or compelled mode for any transactions which are 8, 16, 32, or 64 bytes in length. The *Read/Write Partial* transactions are to transfer 7 bytes or less and are restricted to the compelled mode.

The FB+ is basically comprised of two individual global buses. The `AD[63:0]` bus is a multiplexed address/data 64-bit path that is responsible for all address and data transfers. The second bus is the arbitration bus. Arbitration messages are interrupts and general system information that can be transferred throughout the system in parallel with data bus activity. In addition, this bus can provide arbitration for a bus *master-elect* while another bus device is the current bus master. This provides the ability to hide some of the latency associated with a distributed arbitration protocol for gaining global bus access.

Basic read or write transactions are conducted in three separate phases. The first phase is called *connection phase* and is initiated by the bus master. During this phase the master drives the `AD[63:0]` bus with the address to read from or write to. In addition, signal lines are driven to indicate the phase of the transaction, the transaction type and the style of transfer, packet or compelled. In *data phase*, which is the second phase, data is transferred via packet or compelled mode over the `AD[63:0]` bus. The last phase in the transaction is *disconnection phase* and is used to terminate the FB+ transaction. The master can issue another transaction (*bus park*), or release the bus tenure to the master-elect waiting to carry out a transaction.

Arbitration in FB+ can be initiated any time, regardless of the state of an ongoing bus transaction. The only dependence on the address bus is `AS*` (Address Sync) which indicates to the system that the bus master is terminating its tenure and the bus will be available. Depending on bus traffic, the arbitration latency can be completely hidden.

We use the TI chip-set [21] for our design, which is comprised of three chips, the TFB2010 arbiter, the SN54-FB/SN74FB 2032 competition transceiver, and the SN54FB/SN74FB 2040 TTL-BTL transceiver. The TFB2010 design greatly simplifies the task of system mes-

COMA Transactions		Futurebus+ B-Profile	
Basic	**Variation**	**Transactions**	**Tag[7:0]**
RD	-M Miss (B)	Read Unlocked	0000001
	-U Uncached (P, B)	Read Partial or Read Unlocked	0000000
	-I Invalidate (B)	Read Unlocked	0000010
WR	-M Miss (B)	Read Unlocked	0000100
	-H Hit (P, B)	Write Partial or Write Unlocked	0001000
	-U Uncached (P, B)	Write Partial or Write Unlocked	0000000
REP	-C Copy	Read Unlocked	0010000
	-R Relocate	Write Unlocked	0010000
	-Rp Relocate	Write Unlocked	0100000
TAS	None	Read Unlocked+ Write Partial	1000000

B: Block Transfer
P: Partial Transfer

Table 1: Transaction mapping

sages and FB+ arbitration. Programming and normal control of the arbitration process is accomplished through the *CSR* (Command and Status Register) bus. CS registers inside the TFB2010 may be written to or read from in order to set arbitration priorities, configure operations, send messages, obtain interrupts, and observe the TFB2010 status. The CS registers together with the distributed arbitration are important features that make our implementation efficient. Some backplane buses without such features may necessitate a certain COMA transaction such as replacement to be implemented in more than one bus transactions.

In the remainder of this section, we summarize the implementation of each transaction. Also, we describe how replacement in local memory can be handled with little overhead.

4.2 RD -M, -U, and -I: Read Request

The RD (ReaD request) transaction is made up of three distinct modes of operation. Two of the modes, -M (Miss) and -I (Invalidate), support the DICE architecture while the -U (Uncached) mode helps to maintain 896.2 Profile B compliance, which is necessary to incorporate 'third-vendor' I/O boards.

4.2.1 RD -M

When a read request issued by a processor misses in the local memory, an RD -M transaction will be issued on the global bus. The transaction will always operate on a complete memory block and use the packet mode of data transfer.

4.2.2 RD -U

The RD -U is a read transaction that will not be cached by the recipient of the data. In addition, the slave node supplying data will not alter the coherence state in the local memory. Unlike the RD -M transaction, RD -U can transfer a byte, word, double word, or even multiple blocks of data using the compelled mode or packet mode of data transfer. An uncacheable read transaction helps to meet two of the implementation goals for the DICE project, architecture support and specification adherence. By providing an uncached transaction, a node can conduct transactions to I/O devices and other resources that are not included in the cacheable shared memory space of the system. Also, RD -U provides direct support for memory references signaled as non-cacheable by the CPU. The second goal of adhering to a specification will allow the design to take advantage of industry standard system support devices such as DMA, bus bridges, and networking support.

4.2.3 RD -I

RD -I is one of the transactions unique to a bus-based COMA. In traditional systems, memory recovery and page write-backs to disk are an ongoing process. With respect to main memory storage, these actions are governed solely by the operating system.

In the DICE multiprocessor main memory is not only distributed but also of a cache structure. Consequently, simply altering a page table entry and writing back a "copy" of a page to disk is insufficient to provide data integrity and coherence. For example, if a page is written back to disk and copies are left in the local memories of processing nodes, regardless of what occurs in the L1 and L2 caches, when the page frame is re-allocated by the operating system there will be two different sets of data available. When the RD -I transaction is used to write a page back to disk, the actual page transfer is most commonly handled by a DMA device, independent of the processor(s). During the connection phase of the transaction all nodes snoop the address. Those with SHN coherence state invalidate their copies. The sole node in EXL or SHO state will complete the transaction by first sourcing the RD -I transaction with the requested block then invalidate its own copy. In addition, each node with a valid copy will also invalidate the L2 cache which will in turn invalidate the L1 cache. Although this problem happens also in a traditional SMP with copy-back cache, the problem is more extended in the COMA and fairly complicated due to the relocation process to handle the last memory block problem.

4.3 WR -M, -H, and -U: Write Request

The WR (Write Request) transaction is also comprised of three modes of operation. An uncached (WR -U) operation is again defined to support the B-profile.

4.3.1 WR -M

The DICE architecture assumes a *write-allocate* policy, however, revising this to *no-allocate* or *allocate-on-demand* should not be difficult. To allocate a block to the local memory, WR -M transaction turns to a RD -M transaction with a different `TAG[7:0]`, which signals invalidation of other copies, if they exist in the system. To support write-invalidate policy for coherence, any write reference to a block in the SHN or SHO state will also incur a WR -M transaction. A WR -M transaction is similar to RD -I transaction, but potential initiating source can be different. By implementing WR -M as a simple transaction the latency seen by the initiating CPU and the length of the global bus tenure can be minimized.

4.3.2 WR -H

WR -H is a write partial transaction used to update remote copies for processor write memory references which hit in local memory on the SHN or SHO states. The TAG[7:0] setting separates it from the partial write of the WR -U transaction. In addition, TAG[0] is used to indicate to the mastering node the presence of any remote copies. If the transaction completes and TAG[0] is asserted then there exists at least a shared copy in the system. However, if the transaction completes with TAG[0] unasserted then the mastering node is able to update a SHN copy to EXL. With a write-invalidate policy, the WR -H transaction is not necessary. However, the WR -H transaction can be useful to optimize DICE further than presented here, which is beyond the scope of this paper.

4.3.3 WR -U

The WR -U is a write transaction that will not be cached by any recipient of the data. Similar to RD -U, WR -U can transfer a byte, word, double word, or even multiple blocks of data using the compelled or packet mode of data transfer.

4.4 REP -C, -R, and -Rp: Replacement and Relocation

REP (Replacement) -C (Copy), REP -R (Relocate), and REP -Rp (Relocate page) are related with the replacement protocol described in Section 3.

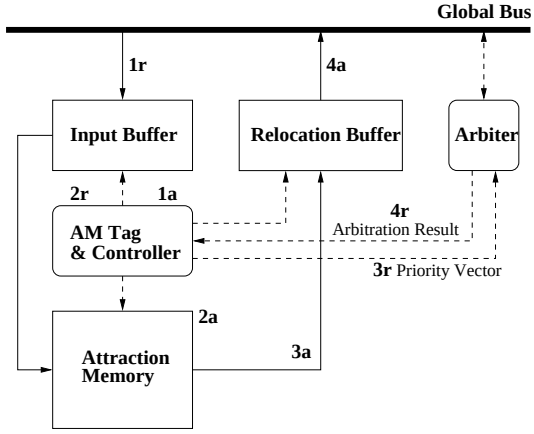

Figure 8: Block relocation mechanism

4.4.1 Relocation mechanism

Figure 8 conceptually demonstrates how this replacement and relocation is handled in a processor node. On a reference miss, the node decides whether relocation is necessary (**1a**). It sends a data request on the bus while fetching the replaced data from the local memory (**2a**). It puts the fetched data into the relocation buffer along with the state (**3a**). Upon the arrival of missing data, it begins the relocate transaction, and the processor now can resume its execution (**4a**).

From the viewpoint of a remote node, when a relocate transaction is seen on the bus, the node buffers the data with its address and state (**1r**). The node looks up the AM state and tag memory to decide its priority in accepting the block it has just received (**2r**). Based on the result of the state and tag look-up, it generates and sends to the arbiter a priority vector, which is the 2-bit priority concatenated with its node ID (**3r**). In case of a tie in the 2-bit priority, the node ID, the lower bits in the vector, will help decide the winner. After arbitration, the result will be passed back to the controller, which will either update the AM and the tag, or discard the buffered data (**4r**). The distributed arbitration determines the unique winner which will accommodate the block, and all other nodes will discard the block, thereby achieving our goal.

4.4.2 REP -C

The REP -C transaction is always performed on non-page fault generated replacements. It is responsible for obtaining a copy of the block that contains the reference missed in the local memory. Although the -C transaction is an unlocked block read as RD -M, following every REP -C, without loss of tenure, is an REP -R transaction.

REP -C is very similar to the transfer mechanism of the RD -M transaction. A global request is issued and the node

with ownership responds by supplying the data. In addition to the data requested, the mastering node also takes over the ownership attribute for the block. This is done to ensure that a location will exist for the relocation transaction following the REP -C. A more complete description and an example of the ownership transfer (and ownership relinquish) is given in [13].

4.4.3 REP -R

When REP -C completes the CPU request can be satisfied and allowed to execute the next instruction. However, the issue of relocating the block which initially occupied the local memory, causing the collision, still remains. The REP -R transaction utilizes the arbitration protocol of the FB+ previously described to accomplish relocation. Without loss of the bus tenure, the REP -R transaction is initiated immediately after a REP -C completes. The transaction is completely controlled by the GBTC (Global Bus Transaction Controller, in Figure 2) and does not involve the LBTC. This allows the LBTC (Local Bus Transaction Controller) to service the CPU for accesses to the local memory.

The GBTC controls the HOST bus and places a block transfer write request to the address of the block needed to be relocated. The global bus interface views this as a packet mode transfer of the number of bytes equal to a block size. Each remote node will search their local memory tags as in previous transactions, however, the response of each node depends on the state of all the blocks in the set to which the address maps. In addition, remote nodes do not simply handshake with the FB+ communication protocol but participate in an arbitration for the block being relocated.

Once nodes have determined their priorities using the scheme outlined in Section 3, each arbitrates using the FB+ arbitration protocol. The arbitration priority of the relocation algorithm is such that master-elect preemption will take place and that only nodes participating in the arbitration have the opportunity to win.

When the arbitration completes, the winner will be the node which takes the block being relocated. The priority of the node winning the arbitration determines what state the block will be placed in. If an INV block or an SHN block not of the same address wins, then the block can be placed in local memory in the EXL state. As in the WR -H transaction, `TAG[0]` is used to remove the ambiguity of relocation. When a SHN state node of the same address wins the arbitration, `TAG[0]` is used to determine if the state should be EXL or SHO.

4.4.4 REP -Rp

In a bus-based COMA multiprocessor, page faults must be managed differently from conventional SMPs. The primary reason is that the local memories of the system are caches to the entire shared address space. Local memories are n-way set-associative and therefore have n locations per node where a page may be located. Also, a page fault in a traditional system generally has no need to alter the location of data already in the system unless memory is full. However, a page fault in a COMA system can result in a significant redistribution of data due to a collision with the incoming page. This can occur if the incoming page maps to a location in local memory which is occupied by block(s) of data in the EXL or SHO state.

When a page fault occurs in DICE, a page frame of memory must be guaranteed to exist which maps to the incoming page. With sufficient unallocated memory, there exists such a page frame [8]. However, guaranteeing available space somewhere in the system does not guarantee available space in a specific node, nor does it guarantee that the available space is contiguous. Since the concept of locality suggests that it would be highly beneficial if the node originating the page fault should also be the recipient of the incoming page [15], clearing specific locations for the incoming page may become necessary. Clearing a page of data in the local memory may only require reserving space if no EXL or SHO attributes currently exist. In the case where all the blocks are not in the INV or SHN state, a relocation transaction becomes necessary for each of those blocks.

When a page fault occurs, the GBTC on the node will begin processing contiguous range of memory associated with the page. The GBTC will go through each block address in the page range and mark INV and SHN copies with the *Occupied* tag status. Blocks in the EXL or SHO state must be relocated as described in Section 4.4.3 with the exception of the relocation buffer, it is not necessary in this case since there is currently not a collision. Once an EXL or SHO block has been relocated the block frame is marked with the Occupied state. When the last block address in the page range reached, the page fault support by the GBTC is complete. Note that blocks coming in from disk and block relocation due to the REP -Rp can be intermixed because of the priority assigned to the REP -Rp transaction and the lower priority of the DMA device when performing the WR -U transaction to move the page into memory. This is fully supported by the FB+ arbitration protocol and the round robin fairness mechanism.

4.5 Synchronization, I/O transactions, and interrupt support

The `TAS` transaction is defined (in Table 1) as a read block followed by a write partial to implement synchronization instructions such as *test-and-set*. The memory location being accessed must remain under the control of a single processor for the duration of the *read-modify-write* cycle. In order to implement this on the TI chip-set two

transactions are required. However, the chip-set provides a means of securing the bus for the duration of both the transactions. A LOCKED* signal input on the HOST bus side of the mastering chip-set can be asserted to ensure that no transfer of tenure occurs between transactions. In addition, remote nodes participating in a locked transaction must also be informed so that local access to the memory between the initial read and subsequent write is not allowed. The TAG[7:0] signal lines are set to inform local nodes that the current transaction is atomic with respect to global memory and a locked transaction on the global bus.

The -U transactions are used in I/O operations as mentioned. However, I/O transactions which involve DMA operations must be treated differently. DMA operations may need to occupy a specific level of priority in the hierarchy of bus transactions in order to ensure that disk accesses and other DMA transfers are allowed adequate bus access. As discussed in Section 4.1, there are many levels of priority available for bus arbitration. Any one of these priorities can be assigned to any bus transaction, and the priorities of specific transactions do not affect the mechanics of the communication protocol. Note that the RD -I transaction is an RD transaction from the DMA where tags are invalidated and the requested block is removed from the system memory.

The FB+ specification and the TI chip-set support various ways to support the system interrupts. General messages can be sent via the bus using the standard unlocked transactions. Interrupts can also be implemented using the arbitration message and 32 dedicated messages. These methods can be combined or used separately to implement global interrupts and interrupts targeted for a specific node.

5 Summary

Although the shared-bus SMP is a widely accepted architecture, its scalability is severely limited due to limited bandwidth the bus can provide. As microprocessors become faster and demand more bandwidth, the shared bus becomes an even more serious bottleneck in such systems. Handling the problem of the shared-bus bottleneck can be done in three complementary approaches. Firstly, a faster and wider bus needs to be developed. Secondly, smart bus protocols such as more aggressive pipelining are needed. Thirdly, memory requests should be serviced locally (or local memory utilization should be high).

We presented a global bus design for DICE, a shared-bus COMA multiprocessor. The main contribution of this paper is in demonstrating the feasibility of an efficient implementation of a bus-based COMA multiprocessor. Our implementation employs the IEEE Futurebus+ (FB+) standard backplane bus and the Texas Instruments chip-set. By using a distributed arbitration mechanism and eight user-definable lines (TAG[7:0]) of the FB+, we demonstrated that all bus transactions necessary for a bus-based COMA multiprocessor can be implemented efficiently with existing bus signals in the standard bus FB+. The implementation presented in this paper can be ported to any backplane bus supporting a snooping coherence protocol with little difficulty. Although replacement in local memory does present unique problems to coherence, our replacement algorithm dynamically chooses an optimal location for data relocation. Using the FB+ distributed arbitration, the overhead associated with finding a relocation node is minimized to only one (long) bus transaction.

With dynamic replication and migration of data through the AMs, a COMA machine seems to be able to provide higher utilization of local memory than is otherwise possible, which can result in low average memory access latency and low network traffic. As the processor technology is progressing much faster than the bus or interconnection network technology, this potential reduction in latency and bandwidth requirement can be a crucial advantage. Considering the benefits of the COMA and the little design complexity it adds to the conventional shared-bus multiprocessor design, a bus-based COMA multiprocessor such as DICE can become a viable candidate for the future shared-bus multiprocessor architecture.

Acknowledgment

The DICE project is supported by funding from Samsung Electronics, Seoul, Korea. Bland Quattlebaum is with the Hewlett-Packard Company, Roseville, CA. We would like to express gratitude to the former and present members of the DICE project: Manu Agarwal, Sujat Jamil, and Jinseok Kong. We thank the anonymous reviewers for their helpful comments.

References

[1] BAER, J.-L. AND WANG, W.-H. "On the Inclusion Property for Multi-level Cache Hierarchies," in *Proceedings of the 15th International Symposium on Computer Architecture*, pp. 73 – 80, 1988.

[2] BERRY, M. *et al.*, "The Perfect Club Benchmark: Effective Performance Evaluation of Supercomputers," in *International Journal of Supercomputing Applications*, Vol. 3, No. 3, 1989.

[3] GALLES, M. AND WILLIAMS, E. "Performance Optimizations, Implementation, and Verification of the SGI Challenge Multiprocessor," in *Proceedings of the 27th International Conference on System Sciences*, Vol. 1, pp. 134 – 143, 1994.

[4] GHARACHORLOO, K., LENOSKI, D., LAUDON, J., GIBBSON, P., GUPTA, A., AND HENNESSY, J.L. "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," in *Proceedings of the 17th International Symposium on Computer Architecture*, pp. 15 – 26, June 1990.

[5] HAGERSTEN, E., LANDIN, A., AND HARIDI, S. "DDM - A Cache-Only Memory Architecture," *IEEE Computer Magazine*, pp. 44 – 54, September 1992.

[6] HENNESSY, J.L. AND PATTERSON, D.A. *Computer Architecture A Quantitative Approach*, Second Ed., Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[7] JAMIL, S. "Block Replacement in Cache-Only Memory Architecture Multiprocessors," *M.S.E.E. Thesis*, Electrical Engineering Department, University of Minnesota, June 1994.

[8] JAMIL, S. AND LEE, G. "Unallocated Memory Space in COMA Multiprocessors," in *Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems*, Orlando, Florida, September 1995.

[9] JOE, T. AND HENNESSY, J.L. "Evaluating the Memory Overhead Required for COMA Architectures," in *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pp. 82 – 93, April 1994.

[10] LAMPORT, L. "How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs," in *IEEE Transactions on Computers*, C-28:9, pp. 241 – 248, September 1979.

[11] LANDIN, A. AND DAHLGREN, F. "Bus-Based COMA – Reducing Traffic in Shared-Bus Multiprocessors," in *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, pp. 95 – 105, February 1996.

[12] LEE, G. AND KONG, J. "Prospects of Distributed Shared Memory for Reducing Global Traffic in Shared-Bus Multiprocessors," in *Proceedings of the 7th IASTED-ISMM International Conference on Parallel and Distributed Computing and Systems*, pp. 63 – 67, Washington, D.C., October 1995.

[13] LEE, G., KONG, J., AND CHO, S. "Coherence and Replacement Protocol for a Bus-Based COMA Multiprocessor DICE," *Technical Report No. 96–008*, Computer Science Department, University of Minnesota, January 1996.

[14] LOVETT, T. AND THAKKAR, S. "The Symmetry Multiprocessor System," in *Proceedings of the 17th International Conference on Parallel Processing*, pp. 303 – 310, August 1988.

[15] MARCHETTI, M., KONTOTHANASSIS, L., BIANCHINI, R., AND SCOTT, M.L. "Using Simple Page Placement Policies to Reduce the Cost of Cache Fills in Coherent Shared-Memory Systems," in *Proceedings of the 9th International Parallel Processing Symposium*, April 1995.

[16] NAYFEHM B.A., OLUKOTUN, K., AND SINGH, J.P. "The Impact of Shared-Cache Clustering in Small-Scale Shared-Memory Multiprocessors," in *Proceedings of the 2nd International Symposium on High-Performance Co mputer Architecture*, pp. 74 – 84, February 1996.

[17] QUATTLEBAUM, B., KINNEY, L., AND LEE, G. "Global Bus Implementation of DICE," *DICE Project Technical Report No. 9*, Electrical Engineering Department, University of Minnesota, January 1994.

[18] QUATTLEBAUM, B., LEE, G., AND KINNEY, L. "Protocol Mapping in Bus-Based COMA Multiprocessors," *DICE Project Technical Report No. 10*, Electrical Engineering Department, University of Minnesota, March 1994.

[19] SINGH, J. P., WEBER, W.-D., AND GUPTA, A. "SPLASH: Stanford Parallel Applications for Shared-Memory," in *Computer Architecture News*, 20(1):5 – 44, March 1992.

[20] WOO, S., OHARA, M., TORRIE, E., SINGH, J., AND GUPTA, A. "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24 – 36, June 1995.

[21] *Futurebus+ Interface Family* (Rev. 5.1), Texas Instruments, Linear Products Div., Dallas, Texas, 1993.

[22] *Microprocessor Systems - Futurebus+ - Logical Protocol Spscifications (ANSI/IEEE Std 896.1 - 1994)*, IEEE, New York, New York, 1994.

[23] *KSR-1 Technical Summary*, Kendall Square Research, Waltham, Massachusetts, 1992.

[24] "Ultra Enterprise X000 Server Family: Architecture and Implementation," *Sun Microsystems*, white paper, April 1996.