DESIGN OF A BUS-BASED SHARED-MEMORY MULTIPROCESSOR FOR COMA LATENCY

Pankaj Sharma*

Naveen Malik*

Naeem Akhtar*

Rahul*

Hardeep Rohilla*

Abstract: *Cache Only Memory Access (COMA) multiprocessors support scalable coherent shared memory with a uniform memory access programming model. The cache-based organization of memory results in long memory access latencies [2].Latency hiding mechanisms can reduce effective memory latency by making data present in a processor's local memory by the time the data is needed. In this paper, we study the effectiveness of latency hiding mechanisms on the KSR2 multiprocessor in improving the performance of three programs. The communication patterns of each program are analyzed and mechanisms for latency hiding are applied. DICE is a shared-bus multiprocessor based on a distributed shared-memory architecture, known as cache-only memory architecture(COMA). [3]Unlike previous COMA proposals for large-scale multiprocessing, DICE utilizes COMA to effectively decrease the speed gap between modem high-performance microprocessors and the bus. DICE tries to optimize 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 of DICE based on the IEEE future bus 1 backplane bus and the Texas Instruments chip-set.[5]*

*Student, CSE, Dronachraya College of Engineering, Gurgaon

1 INTRODUCTION:

Cache only memory architecture (**COMA**) is a computer memory organization for use in multiprocessors in which the local memories (typically DRAM) at each node are used as cache. This is in contrast to using the local memories as actual main memory, as in Non Uniform Memory Access(NUMA) organizations. In NUMA, each address in the global address space is typically assigned a fixed home node. When processors access some data, a copy is made in their local cache, but space remains allocated in the home node. Instead, with COMA, there is no home. An access from a remote node may cause that data to migrate. Compared to NUMA, this reduces the number of redundant copies and may allow more efficient use of the memory resources. On the other hand, it raises problems of how to find a particular data (there is no longer a home node) and what to do if a local memory fills up (migrating some data into the local memory then needs to evict some other data, which doesn't have a home to go to). Hardware memory coherence mechanisms are typically used to implement the migration. Shared-bus symmetric multiprocessors (SMPs) have been widely used as a computing vehicle for small-scale multiprocessing [1, 2]. As microprocessors become faster and demand more bandwidth, however, the already limited scalability of the bus decreases further, and the ill-effect of a cache miss penalty becomes worse. Even with clustering of several processors per processor board, the effective machine size for shared-bus multiprocessors is fairly limited. Moreover, a cache miss can cost up to a few hundred processor cycles for recent high-performance microprocessors. To bridge the speed gap between high performance microprocessors and a backplane bus, it is important to reduce global bus traffic by increasing 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 cache-only memory architecture (COMA) to bridge the gap. COMA improves the utilization of local memory by decoupling the address of a datum from its physical location, allowing the data to migrate and replicate 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 [3].

2. BUS-BASED COMA ARCHITECTURE:

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.

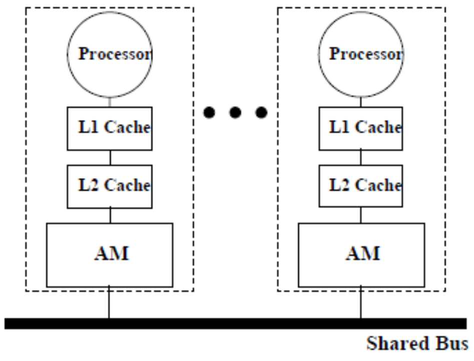


Figure 1: A bus-based COMA multiprocessor

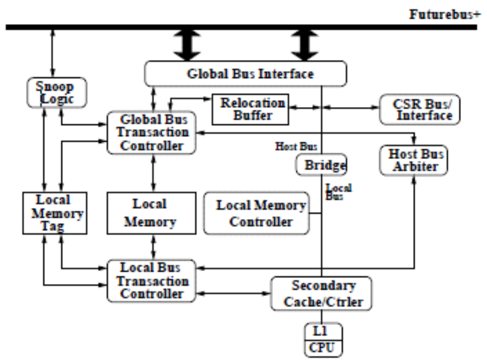


Figure 2: Block diagram of a DICE node

Figs. 3 shows the bus utilization and traffic rate per reference for the studied architectures, respectively. For the nine programs from the Perfect Club Benchmark [18], our simulation results show that DICE can reduce bus traffic significantly. DICE, though, generated slightly more traffic for replacement and coherence for some programs. A recent study on a bus-based COMA multiprocessor reports a similarly significant reduction in bus traffic [19]: a traffic reduction of up to 70%, with an average of 46%, for the six SPLASH benchmark programs.

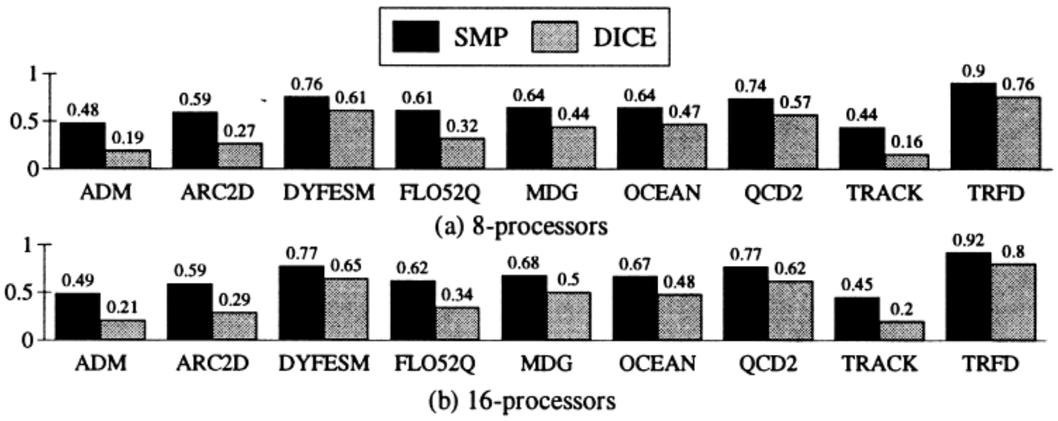


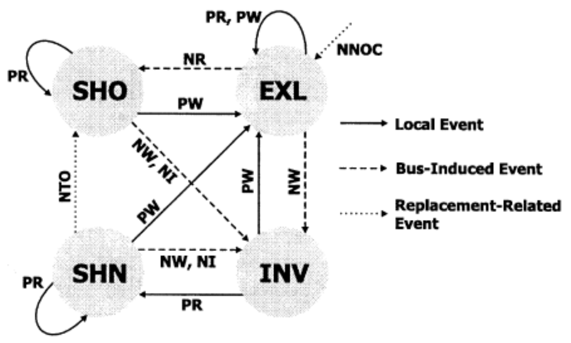
Fig. 3. Global bus utilization (memory pressure ≈ 60%).

REPLACEMENT AND COHERENCE:

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.

G. Lee et al. / Microprocessors and Microsystems 22 (1999) 403–411



DICE write-invalidate coherence protocol (PR: processor read; PW: processor write; NR: network read; NW: network write; NI: network invalidation;

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 Future bus+ standard bus. The implementation presented here is one of many possible implementations. Although the design described in [17, 18] uses a *write-update* policy, our discussion is limited to the one with a *write-invalidate* policy.

Table 1
Transaction mapping. B: block transfer. P: partial transfer

COMA transactions		Futurebus + B-profile	
Basic	Variation	Transactions	TAG[7:0]
	− M miss (B)	Read unlocked	0000001
RD	− U uncached (P, B)	Read partial or read unlocked	0000000
	− I invalidate (B)	Read unlocked	0000010
	− M miss (B)	Read unlocked	0000100
WR	− H hit (P, B)	Write partial or write unlocked	0001000
	− U uncached (P, B)	Write partial or write unlocked	0000000
	− C copy	Read unlocked	0010000
REP	− R relocate − Rp relocate	Write unlockedWrite unlocked	00100000100000
TAS	None	Read unlocked + write partial	1000000

4.1 FB+ background

We chose the Future bus+ (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.

4.2. RD 2 M, 2 U and 2 I: read request

The RD (Read request) transaction is made up of three distinct modes of operation. Two of the modes, 2 M (miss) and 2 I (invalidate), support the DICE architecture while the 2 U (uncached) mode helps to maintain 896.2 profile B compliance, which is necessary to incorporate 'third 2 vendor' I/O boards.

4.2.1. RD 2 M

When a read request issued by a processor misses in the local memory, an RD 2 M transaction will be issued on the global bus. The transaction will always operate on a complete memory block and use the packet mode.

4.2.2. RD 2 U

The RD 2 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. An un cacheable read transaction helps to meet two of the implementation goals for the DICE project: architecture support and specification adherence. By providing an un cached 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 2 U provides direct support for memory references signaled as non 2 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 2 I

RD 2 I is one of the transactions unique to a bus 2 based COMA multiprocessor. In traditional systems, memory recovery and page write 2 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 2 allocated by the operating system there will be two different sets of data available.

When the RD 2 I transaction is used to write a page cached by any recipient of the data. Similar to RD 2 U, WR 2 U can transfer a byte, word, double word, or even multiple blocks of data using the compelled or packet mode of data transfer.

4.4.1 Relocation mechanism

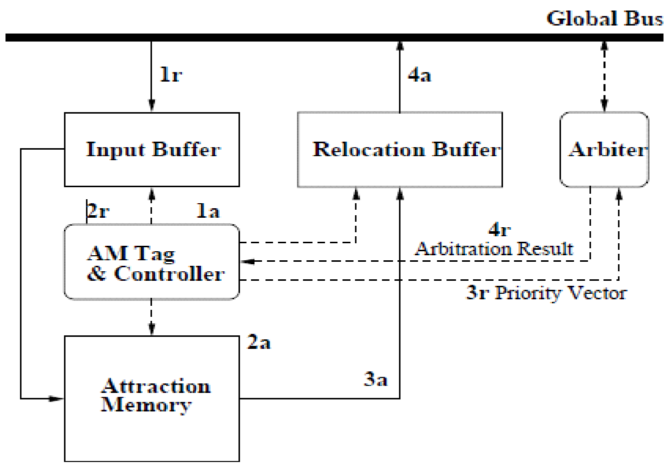


Figure 8: Block relocation mechanism

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 pre emption 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 2 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 2 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 access 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. The FB 1 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. 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. 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). With dynamic replication and migration of data through the AMs, a COMA machine is expected 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 technology, this potential reduction in latency and bandwidth requirement can be a crucial advantage. Considering the benefits of COMA and the moderate 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.

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] B.W. Quattlebaum, L.L. Kinney, G. Lee, Global bus implementation of DICE, DICE project technical report no. 9, Department of Electrical Engineering, University of Minnesota, January 1994.

[22] B.W. Quattlebaum, G. Lee, L.L. Kinney, Protocol mapping in busbased COMA multiprocessors, DICE project technical report no. 10, Department of Electrical Engineering, University of Minnesota, March 1994.

[23] Microprocessor Systems — Futurebus 1 — Logical Protocol Specifications (ANSI/IEEE Std 896.1 — 1994), IEEE, New York, 1994.

[24] Futurebus 1 Interface Family (Rev. 5.1), Texas Instruments, Linear Products Division, Dallas, Texas, 1993.

[25] M. Marchetti, L. Kontothanassis, R. Bianchini, M.L. Scott, Using simple page placement policies to reduce the cost of cache fills in coherent shared memory systems, Proceedings of the 9th International Parallel Processing Symposium, April 1995.

[26] G. Lee, B. Quattlebaum, S. Cho, L. Kinney, Global bus design of a bus-based COMA multiprocessor DICE, Proceedings of the IEEE International Conference on Computer Design, October 1996, pp. 231–240.

[27] T. Lovett, S. Thakkar, The symmetry multiprocessor system, Proceedings of the 17th International Conference on Parallel Processing, August 1988, pp. 303–310.

[28] M Galles , E. Williams, Performance optimizations, implementation and verification of the SGI challenge multiprocessor, Proceedings of the 27th Hawaiian International Conference on System Sciences 1 (1994) 134–143.

[29] E. Hagersten, A. Landin, S. Haridi, DDM — a cache-only memory architecture, IEEE Computer Magazine September (1992) 44–54.[30KSR-1 Technical Summary, Kendall Square Research, Waltham, MA, 1992.

[31] G. Lee, Block replacement method in cache only memory architecture multiprocessor, US patent no. 5 692 149.[32] S. Jamil, Block Replacement in Cache-Only Memory Architecture Multiprocessors, M.S.E.E. Thesis, Department of Electrical Engineering, University of Minnesota, June 1994.

[32] T. Joe, J.L. Hennessy, Evaluating the memory overhead required for COMA architectures, Proceedings of the 21st International Symposium on Computer Architecture, April 1994, pp. 82–93.

[33] Gyungho Lee, Bland Quattlebaum, Sangyeun Cho, and Larry Kinney Dept. of Electrical Engineering Dept. of Computer Science University of Minnesota Minneapolis, MN 55455

[34] Gyungho Leea,*, Bland W. Quattlebaumb, Sangyeun Choc, Larry L. Kinneyd a Division of Engineering, University of Texas, San Antonio, TX 78249-0665, USA bHewlett Packard Company, Roseville, CA 95747-6588, USA cDepartment of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA dDepartment of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA Received 27 February 1998; received in revised form 16 June 1998; accepted 18 June 1998