ELSEVIER

Microprocessors and Microsystems 21 (1998) 313–318

MICROPROCESSORS AND
MICROSYSTEMS

On timing constraints of snooping in a bus-based COMA multiprocessor[1]

Sangyeun Cho[a],*, Jinseok Kong[a], Gyungho Lee[b]

[a]*Dept. of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA*
[b]*Division of Engineering, University of Texas at San Antonio, San Antonio, TX 78249-0665, USA*

Received 25 March 1997; received in revised form 5 August 1997; accepted 2 September 1997

Abstract

Cache only memory architecture has the potential to decrease global bus traffic in shared-bus multiprocessors, thereby reducing the speed gap between modern microprocessors and global backplane bus systems. However, the (huge) size of attraction memory (AM) in each processor node makes it difficult to properly match the access time of its state and tag storage to the bus cycle. This becomes a serious burden in efficient snooping, much more than in conventional shared-bus multiprocessors, especially when a high bus clock frequency is used. In this paper, we propose a scheme to relax the timing constraints of snooping in a bus-based COMA multiprocessor, which allows an efficient design of a global bus protocol, and a cost-effective implementation of the overall system by using slower and cheaper memory for the state and tag storage of AM. © 1998 Elsevier Science B.V.

Keywords: Cache coherence; Cache only memory architecture; Shared-bus multiprocessor; Snooping

1. Introduction

Cache-coherent shared-bus SMPs (symmetric multi-processors) such as Sequent Symmetry [1] or SGI Challenge [2] represent the mainstream of accepted and commercially viable computer systems. However, as microprocessors become faster and demand more bandwidth, the already limited scalability of a bus decreases even further, and the ill effects of a cache miss penalty become even worse. The effective machine size for a shared-bus SMP is fairly limited, typically less than twenty processors, and 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 back-plane bus, it is very 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.

Cache only memory architecture (COMA) seems to be a viable candidate to effectively tackle the shared-bus bottleneck. Recent studies [3–6] have shown that bus-based COMA multiprocessors can reduce the global bus traffic by a large amount. With 16 processors, a traffic reduction of up to 65% with an average of 28% was observed in [4], and up to 70% with an average of 46% in [5]. Considering current microprocessors that need aggressive data bandwidth, the bus-based COMA multiprocessor is of increasing interest.

With a high bus clock frequency used in recent machines, it is important to design snooping circuitry that can operate fast enough to match the bus clock frequency. Due to the large size of the state and tag storage of the attraction memory (AM) in COMA, it takes longer to access the AM for snooping, i.e., extra time is consumed for longer address decoding and multiplexing due to high set-associativity [7]. Thus, it is even harder for a snooper to keep up with the high clock frequency of a modern high-performance bus in a COMA machine. For example, SGI Challenge uses 47.6 MHz for bus clocking [2], and more recent SMP servers from Sun Microsystems use 83.3 MHz [8], which implies that the snooper has a shorter time to look up the state and tag storage, or needs more bus cycles than before. Moreover, fast state and tag storage to match short clock cycles can be a significant overhead in terms of cost due to its large size. Accordingly, it will be beneficial to develop techniques to relax the dependence a snooper may have on the global bus clock, so that a high clock frequency can be used and the system cost can be kept low as well.

In this paper, we describe a scheme to relax timing constraints for the snooping in a bus-based COMA

* Corresponding author.
[1] This work was supported by funding from Samsung Electronics, Seoul, Korea.

0141-9331/98/$19.00 © 1998 Elsevier Science B.V. All rights reserved
PII S0141-9331(97)00055-0

multiprocessor by utilizing properties of the COMA cache coherence protocol and a relaxed memory consistency model [9]. We present a snoopy scheme with relaxed timing constraints in the context of the SGI POWERpath-2 bus [2] and the DICE, a bus-based COMA multiprocessor [4,6].

This paper is organized as follows. Section 2 briefly introduces a bus-based COMA multiprocessor as a background necessary for our discussions. Section 3 discusses a mechanism to relax the timing constraints of snooping in a bus-based COMA multiprocessor, and Section 4 will summarize the paper.

2. Background

2.1. Bus-based COMA multiprocessor

2.1.1. Overall architecture

Fig. 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 attraction memory (AM), which is organized and managed as a cache to the global address space. The AM tag, which includes 'state' information, is duplicated so that local tag access and global bus snooping will not conflict too often. The inclusion property [10] is maintained in the memory hierarchy. Lee et al. [6] describe a global bus design of a bus-based COMA multiprocessor in the context of the Futurebus+ standard bus specification, and shows that it is feasible to design an efficient global bus for a COMA multiprocessor with minor modifications to existing systems. A more detailed description of the cache coherence mechanism and architecture of a bus-based COMA multiprocessor can be found in [4,5].

2.1.2. Attraction memory (AM)

Unlike the traditional main memory, AM is organized as a cache to the lower-level memory hierarchy. This unconventional structure of the AM allows dynamic replication and migration of memory blocks as they are accessed by processors. It is important to have some of the physical storage space in the AM left unallocated, i.e. not utilized as a part of the physical address space, in order to aid efficient data replication. For example, a COMA machine of 16 nodes with 64 MBytes of AM per node may have only 512 MBytes for its physical address space, leaving 512 MBytes unallocated. Proper reservation of the unallocated space needs to be performed in consideration with the set-associativity of the AM, and can be handled by the operating system with appropriate hardware support [7,11]. For AM coherence, each memory block is associated with state information, and an AM coherence protocol (such as the one in Section 3.1) is necessary.

To see the overhead due to the size of the state and tag storage for AM, let us take an example of a machine with 16 nodes, each of which has 256 MBytes of 4-way set-associative AM, with an overall AM size of 4 GBytes. The size (number of bits) of the state and tag storage is given by (no. of blocks) × ((no. of tag bits per block) + (no. of state bits per block)). Assuming a 128-Byte block size, a 40-bit physical address, and a 2-bit state, this amounts to 32 Mbits (256M/128 × (40-7-log((256M/4)/128) + 2)) per node, which is roughly 1.6% of the AM size. If the storage is duplicated to efficiently support snooping, the cost should be doubled. It is economically unwise to provide very fast SRAMs for this large size of the state and tag storage. However, with slow state and tag storage, a fast bus cycle does not translate into better performance.

2.1.3. Potential performance

Since global bus traffic can become a severe performance bottleneck in a shared-bus SMP [12], the capability of COMA to increase the local memory utilization and thus decrease the global bus traffic can help the system achieve higher performance, as reported by some recent papers [3,5]. For a set of FORTRAN benchmark programs, [3] reports that a traffic reduction of up to 65% with an average of 28% was achieved. Similarly, [5] reports that up to 70% with an average of 46% of bus traffic reduction was observed for a set of benchmark programs written in C. In terms of the execution time, [5] found execution time reductions of up to 59% with an average of 32%. These results reveal the potential of COMA as a viable memory architecture for a shared-bus SMP.

2.2. Revisiting snooping

To avoid the stale data problem (or cache coherence problem) in a cache-coherent shared-bus SMP, each processor node with private cache memory snoops all bus transactions (memory request and invalidation), either to provide sharing information and provide data if needed, or to change the state of its local copy. Sharing information is propagated by asserting a special bus line (e.g., shared*) to inform the requesting node of which state among 'shared' and 'exclusive' the missed block needs to be loaded in. For correct operation of the system, a read request is retried if any remote node was unable to respond with the information

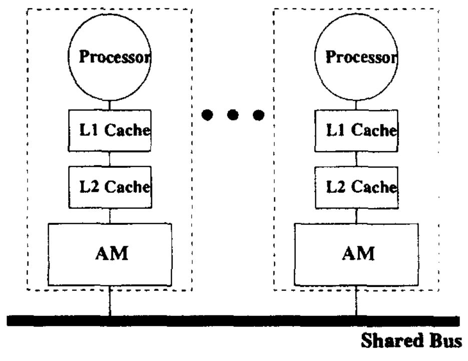


Fig. 1. A bus-based COMA multiprocessor.

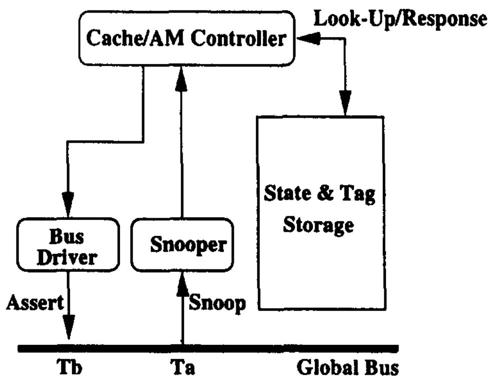


Fig. 2. Snooping activity.

in time. Snooping in practice is mostly under pressure of tight timing and often wastes bus cycles due to late snoop acknowledgments from other nodes.

Fig. 2 shows the operations done by a bus snooper. At time T_a, the snooper latches the address of a memory request. It then looks up the state and tag storage of its cache memory or AM, gets the result, and asserts the shared* line and other acknowledgments at T_b accordingly. We define the maximum valid $(T_b - T_a)$ as snoop turnaround time (or STT). A long STT from erroneous conditions will lead to a bus transaction retry.

Now consider Fig. 3, where a read request transaction in the SGI Challenge SMP [2] is depicted. It consists of 5 phases: arbitration, resolution, address, decode, and acknowledgment. Address is put on the bus in third cycle, when each processor node latches the address and starts looking up the state and tag storage of its cache. T_c is the time when the address of the request is available for latching by processor nodes. T_d is the time when each processor node should drive control lines so that the requesting node will collect the sharing information and decide whether or not to retry the transaction. We define $(T_d - T_c)$ as snoop cycle time (SCT), which is related to the clock frequency and the number of cycles per transaction. About 30–35 ns of SCT is given to each node in the Challenge SMP. Clearly, the STT should be smaller than the SCT, or $(T_b - T_a) < (T_d - T_c)$. In a traditional shared-bus SMP, the snooper can manage to meet this requirement with fast SRAMs of 5–10 ns access/cycle time currently available [13].

However, in a COMA multiprocessor, we employ a huge local memory (AM), and it takes longer to retrieve a datum from the state and tag storage compared to that of a small cache, due to the extra time consumed for address decoding and multiplexing. This will increase the STT, forcing the SCT to be longer, which may decrease the clock frequency or lengthen the number of bus cycles in a transaction. Since the bandwidth of a bus B is defined as

$$B = D \times F \times E$$

where D is the bus width for data, F the bus clock frequency, and E the efficiency of the bus protocol, the bandwidth may become smaller in proportion to the amount of decrease in F or E. This creates the snooping problem in a bus-based COMA multiprocessor: the STT has to be fast enough not to limit the bus transaction frequency, and yet the state and tag storage has to be cheap enough not to increase the system cost unnecessarily [5,6].

A previous work [5] uses a buffer called IB fifo and a cache to the STM (state and tag memory) called TraSM (transient state memory) to handle this problem. Their technique buffers the bus requests in the IB fifo, and stores transient state information about ongoing accesses in TraSM, in order to control the system cost (by adopting hierarchical tag memory—fast TraSM and slow STM) and allow for the access time variation with buffering. In this paper, we utilize the properties of a COMA coherence protocol and a relaxed memory model [9] to tackle the problem without introducing an additional cache.

3. Relaxing snooping

3.1. Cache coherence in a bus-based COMA multiprocessor

We start this section by introducing a cache coherence protocol for a bus-based COMA multiprocessor DICE [4,6]. Fig. 4 shows the four-state write-invalidate coherence protocol for the DICE multiprocessor, which utilizes 'ownership' to reduce coherence overhead. 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

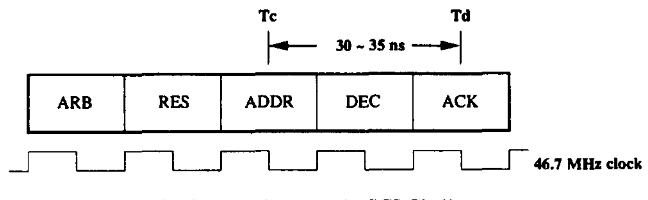


Fig. 3. A read request in SGI Challenge.

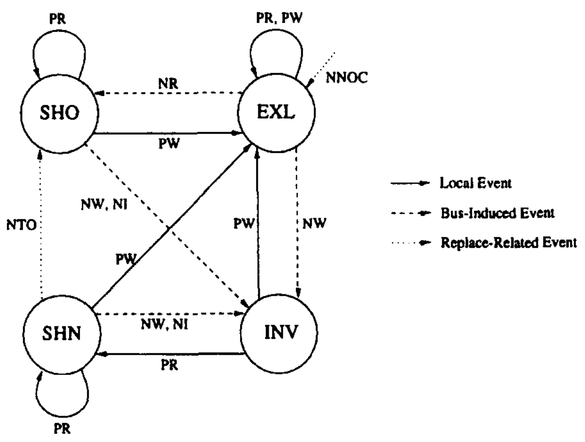


Fig. 4. The DICE write-invalidate coherence protocol (PR: Processor Read, PW: Processor Write, NR: Network Read, NW: Network Write, NI: Network Invalidation, NTO: Network Transfer of Ownership, NNOC: Network No Other Copy).

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. There is no need for 'modified' state in the protocol, since there is no distinction between an EXL block and a modified block in that they are the only copy of a block in the system, carrying the ownership of the block. Note that the 'modified' state is used for write-back on replacement in traditional memory hierarchy.

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. The replacement problem in COMA is not addressed in this paper due to limited space. Lee et al. [4] describes in detail an efficient coherence and replacement protocol for the DICE multiprocessor.

3.2. Mechanism of relaxation

The above-presented DICE coherence protocol has two desirable properties that can help relax the timing constraints of snooping. The first property is that a read miss always brings a block in the SHN state if there was no page fault. This is because the owner of the block is a processor node, not the main memory (there is no central main memory in COMA), and after a read miss is satisfied, the degree of sharing for the block becomes at least two. The property obviates the need for each node to look up its state and tag storage and give back the sharing information in a strictly synchronous manner as in Fig. 3. Instead, each node can buffer the request to perform necessary coherence actions at a later time.

The second property is that there is only one owner of an AM block. The property uniquely determines which node is responsible for providing data to a requesting node. In SGI Challenge, when there is an ambiguity of which, among caches and the main memory, provides the data, the cache suppresses the main memory by asserting the inhibit* signal line and provides the block when it is known that the main memory has stale data [2]. The second property removes the complexity of driving inhibit* signals, making the AM/Bus controller design simpler.

Based on these properties, a snooping mechanism and request handling operations are shown in Figs. 5 and 6, respectively. Note that STT is not dependent on the state and tag storage access time any more but on the queue insertion operation (see Snoop() in Fig. 6). Each

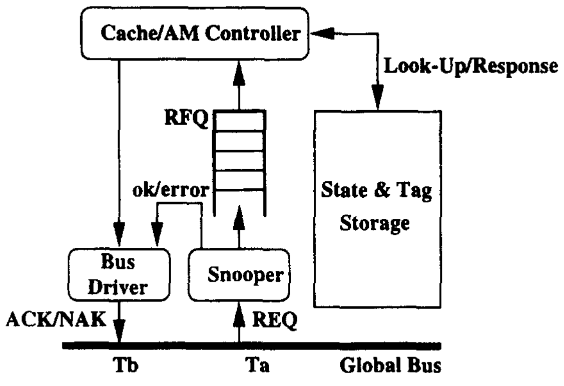


Fig. 5. Mechanism for relaxed snooping.

request is stored in a small and fast FIFO queue called Request FIFO Queue (RFQ), and if RFQ is full, a NAK is returned by the bus driver, so that the request can be retried later.

Although timings may vary, the requests (including coherence requests) are seen and processed by each node in the same order (the bus serializes requests and FIFO keeps the order). Since coherence requests are observed in the same sequence by every node, a program will run correctly on the relaxed snooping mechanism [9].

FIFO_Handler() in Fig. 6 shows how a request on top of RFQ is processed. For a read request, the handler needs to look up the state and tag storage with the address of the request first. Upon deciding whether to provide data based on the ownership information, the handler can initiate data fetching and change the block state if needed, or ignore the rest of the operations. Thus, the ownership information filters out unnecessary operations due to read requests. Note that in the RFQ there cannot be multiple entries of requests targeting the same block which the node owns, since pending read buffer (read resources in SGI Challenge [2], not shown in the figures) forbids such a request from coming out to the bus until the owner node has finished answering to the previous request. A write request is similarly processed when it appears on the top of the RFQ and so is an invalidation request.

By the mechanism presented in this section, a read request, the most common bus transaction, can be done independently of the state and tag storage access time, thus having possibly shorter latency than a transaction in conventional SMPs. Thus, we may lower the system cost by using cheaper and slower memory for the state and tag storage without causing significant performance loss. However, it is needed to quantitatively study the performance impact of this mechanism on a realistic bus-based COMA multiprocessor model.

4. Conclusion

In this paper we examined the snooping problem in a bus-based COMA multiprocessor and presented a mechanism to

```
Snoop (input)                              // called on input
{
  if (input == RELOCATE)
    Relocate ();                           // handler for relocation
  else if (Put_FIFO (input) == SUCCESS)// buffer not full ?
         ACK ();                           // quick answer ack
       else NAK ();                        // quick answer nak
}

FIFO_Handler ()
{
  while (1) {
    switch (top) {                         // top element of FIFO
      case READ_REQ:                       // on a read request
        st = LookUp (top.addr);            // state and tag look-up
        if (st == DONT_HAVE) break;        // don't bother
        if (st == EXL or SHO)              // if owner
          Provide_Data (top.addr);
        else { remove top; break; }        // none of my business
        st' = FSM (Read_Req, st);          // st' is the new state
        if (st != st')
          Set_State (top.addr, st');
        Remove_Top ();
        break;

      case WRITE_REQ:                      // on a write request
        st = LookUp (top.addr);            // state and tag look-up
        if (st == DONT_HAVE) break;        // don't bother
        if (st == EXL or SHO)              // if owner
          Provide_Data (top.addr);
        Set_State (top.addr, INV);         // invalidate
        Remove_Top ();
        break;

      case INV_REQ:                        // on invalidation request
        Set_State (top.addr, INV);         // invalidate
        Remove_Top ();
        break;

      default:
        break;
    }
  }
}
```

Fig. 6. Operations of snooping and request handling.

ease the burden of the snooping constraints imposed on the bus and the AM controller design. By utilizing the properties of the COMA cache coherence protocol and relaxed memory consistency models, the burden of strictly synchronized snooping can be relaxed. This relaxation can be valuable in case the snooper has very tight timing, causing some wasted cycles due to late snoop acknowledgments.

Relaxation of snooping constraints can contribute to shorter latency of common bus transactions and decreasing the system cost with careful choice in system components. The proposed relaxation of the bus snooping guarantees that a program will run correctly on a bus-based COMA multiprocessor as long as the program obeys a set of rules for synchronization as relaxed memory consistency models specify. It seems interesting to study the effects of the mechanism on the overall performance of a bus-based COMA multiprocessor executing real applications.

Acknowledgements

We thank the anonymous referees for their constructive comments and encouragements. A version of this paper appeared in the Proc. of IASTED International Conference on Parallel and Distributed Computing and Systems, Chicago, IL, October 1996.

References

[1] T. Lovett, S. Thakkar, The symmetry multiprocessor system, in: Proc. of the 17th Int. Conference on Parallel Processing, August 1988, pp. 303-310.

[2] M. Galles, E. Williams, Performance optimizations, implementation, and verification of the SGI Challenge multiprocessor, in: Proc. of the 27th Int. Conference on System Sciences, vol. 1, 1994, pp. 134-143.

[3] G. Lee, J. Kong, Prospects of distributed shared memory for reducing global traffic in shared-bus multiprocessors, in: Proc. of the 7th IASTED-ISMM Int. Conference on Parallel and Distributed Computing and Systems, Washington D.C., October 1995, pp. 63-67.

[4] G. Lee, J. Kong, S. Cho, Coherence and replacement protocol for a bus-based COMA multiprocessor DICE, Technical Report No. 96-008, Dept. of Computer Science, Univ. of Minnesota, January 1996.

[5] A. Landin, F. Dahlgren, Bus-based COMA—reducing traffic in shared-bus multiprocessors, in: Proc. of the 2nd Int. Symposium on High-Performance Computer Architecture, San Jose, CA, February 1996, pp. 85-105.

[6] G. Lee, B. Quattlebaum, S. Cho, L. Kinney, Global bus design of a bus-based COMA multiprocessor DICE, in: Proc. of the IEEE Int. Conference on Computer Design, Austin, TX, October 1996, pp. 231-240.

[7] S. Jamil, G. Lee, Unallocated memory space in COMA multiprocessors, in: Proc. of the 8th Int. Conference on Parallel and Distributed Computing Systems, Orlando, FL, September 1995.

[8] Ultra Enterprise X000 Server Family: Architecture and Implementation, Sun Microsystems, White Paper, April 1996.

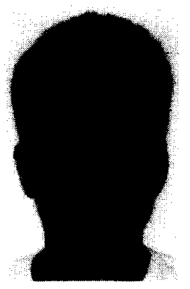
[9] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbson, A. Gupta, J.L. Hennessy, Memory consistency and event ordering in scalable shared-memory multiprocessors, in: Proc. of the 17th Int. Symposium on Computer Architecture, June 1990, pp. 15-26.

[10] J.-L. Baer, W.-H. Wang, Architectural choices for multi-level cache hierarchies, in: Proc. of the 16th Int. Conference on Parallel Processing, 1987, pp. 258-261.

[11] T. Joe, J. Hennessy, Evaluating the memory overhead required for COMA architectures, in: Proc. of the 21st Annual International Symposium on Computer Architecture, April 1994, pp. 82-93.

[12] J.L. Hennessy, D.A. Patterson, Computer Architecture A Quantitative Approach, 2nd ed., Morgan Kaufmann, San Francisco, CA, 1996.

[13] Computer Design, Pennwell, Vol. 35, No. 3, February 1996.



Sangyeun Cho is a PhD student in Computer Science and Engineering at the University of Minnesota where he is a Graduate Research Assistant for the Agassiz project. His current research focuses on high-performance microprocessors, shared-memory multiprocessors, compiler techniques for such architectures, and their performance evaluation. Cho received a BS in Computer Engineering from Seoul National University, Seoul, Korea in 1994 and an MS in Computer Science from the University of Minnesota in 1996. He is a student member of the ACM, the IEEE, and the IEEE Computer Society.

Jinseok Kong is a PhD candidate in Computer Science at the University of Minnesota. His research interest is computer architecture. He received a BS and an MS in Computer Science from Seoul National University, Seoul, Korea.

Gyungho Lee has been with the faculty of the Division of Engineering at the University of Texas at San Antonio since 1996. Prior to joining the University of Texas at San Antonio, he was an Assistant Professor in the Dept. of Electrical Engineering at the University of Minnesota from 1988 to 1996 and an Assistant Professor at the University of SW Louisiana in Lafayette from 1986 to 1988. While he was on a leave of absence from the University of Minnesota from 1990 to 1992, he worked as the principal architect of SSM7000, the first commercial shared-memory multiprocessor in Korea, which is currently being marketed by Samsung Electronics. He was responsible for the design of coherence protocol and two-level cache memory in addition to the overall architecture of SSM7000. Dr. Lee's research interests are in high-speed packet switch architecture for multiprocessor interconnection and ATM network, multiprocessor memory architectures, and compiler support for high-performance computing. Dr. Lee is a senior member of the IEEE, and currently serves on the editorial board for the International Journal of Computer and Software Engineering.