Scalable Shared-Memory Multiprocessor Architectures

New coherence schemes scale beyond single-bus-based, shared-memory architectures. This report describes three research efforts: one multiple-bus-based and two directory-based schemes.

Introduction

Shreekant Thakkar, Sequent Computer Systems
Michel Dubois, University of Southern California
Anthony T. Laundrie and Gurindar S. Sohi, University of Wisconsin-Madison

There are two forms of shared-memory multiprocessor architectures: bus-based systems such as Sequent's Symmetry, Encore's Multimax, SGI's Iris, and Stardent's Titan; and switching network-based systems such as BBN's Butterfly, IBM's RP3, and New York University's Ultra. Because of the efficiency and ease of the shared-memory programming model, these machines are more popular for parallel programming than distributed multiprocessors such as NCube or Intel's iPSC. They also excel in multiprogramming throughput-oriented environments. Although the number of processors on a single-bus-based shared-memory multiprocessor is limited by the bus bandwidth, large caches with efficient coherence and bus protocols allow scaling to a moderate number of processors (for example, 30 on Sequent's Symmetry).

Bus-based shared-memory systems use the bus as a broadcast medium to maintain coherency; all the processors "snoop" on the bus to maintain coherent information in the caches. The protocols require the data in other caches to be invalidated or updated on a write by a processor if multiple copies of the modified data exist. The bus provides free broadcast capability, but this feature also limits its bandwidth.

New coherence schemes that scale beyond single-bus-based, shared-memory architectures are being proposed now that the cost of high-performance interconnections is dropping. Current research efforts include directory-based schemes and multiple-bus-based schemes.

Directory-based schemes. Directory-based schemes can be classified as centralized or distributed. Both categories support local caches to improve processor performance and reduce traffic in the interconnection. The following "coherence properties" form the basis for most of these schemes:

- *Sharing readers.* Identical copies of a block of data may be present in several caches. These caches are called readers.
- *Exclusive owners.* Only one cache at a time may have permission to write to a block of data. This cache is called the owner.
- *Reader invalidates.* Before a cache can gain permission to write to a block (that is, become the owner), all readers must be notified to invalidate their copies.
- *Accounting.* For each block address, the identity of all readers is somehow stored in a memory-resident directory.

Presence flags. One cache-coherence scheme, proposed by Censier and Feautrier[1] in 1978, uses presence flags. In each

0018-9162/90/0600-0071$01.00 © 1990 IEEE

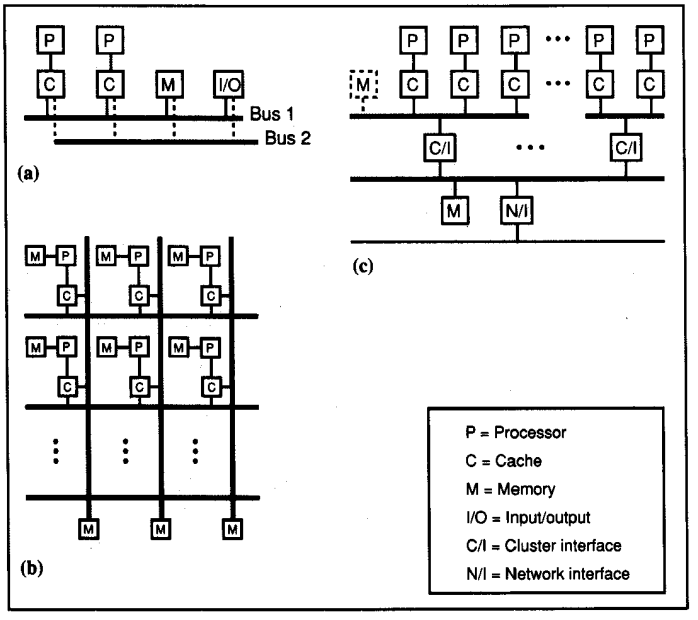

Figure 1. Extension of single-bus architectures to multiple-bus architectures: (a) one dimension; (b) two dimensions; (c) hierarchy.

memory module, every data block is appended with a single state bit followed by one presence flag per cache. Each presence flag is set whenever the corresponding cache reads the block. As a result, invalidation messages need only be sent to those caches whose presence bits are set.

In the presence-flag solution, unfortunately, the size of each memory tag grows linearly with the number of caches, making the scheme unscalable. The tag will be at least N bits, where N is the number of caches in the system. There may also be other bits stored in the directory to indicate line state.

Variations of this scheme use a broadcast mechanism to reduce the number of bits required in the directories. However, this introduces extra traffic in the interconnection and may degrade system performance.

In the central-directory scheme with presence flags, a cache miss is serviced by checking the directory to see if the block is dirty in another cache. When necessary, consistency is maintained by copying the dirty block back to the memory before supplying the data. The reply is thus serialized through the main memory. To ensure correct operation, the directory controller must lock the memory line until the write-back signal is received from the cache with the dirty block. Write misses generate additional invalidate messages for all caches that have clean copies of the data. Invalidate-acknowledgments must be received before a reply can be sent to the requesting cache. Note that the relevant line is locked while this is being done. Requests that arrive while a line is locked must be either buffered at the directory or bounced back to the source to be reissued at a later time. This may cause a loss in performance.

The performance of presence-flag schemes is limited by conflicts in accessing the main memory directory. The main memory and the tags can be distributed to improve the main memory's performance. However, the serialization of responses through the main memory and the locking of lines by the directory controller affect the performance of these cache-coherence schemes.

B pointers. Another alternative, being pursued by Agarwal et al.[2] and by Weber and Gupta,[3] requires each block to have a smaller array of B pointers instead of the large array of presence bits. Some studies of application programs[2-4] suggest that, because of the parallel programming model used, a low value for B (perhaps 1 or 2) might be sufficient. Since each shared data structure is protected by a synchronization variable (lock), only the lock — not the shared data structure — is heavily contested in medium- to large-grain parallel applications. Thus, the shared data only moves from one cache to another during computation, and only synchronization can cause invalidation in multiple caches. If B is small and the data is heavily shared, processors can thrash on the heavily shared data blocks. If B is large, the memory requirements are worse than for the presence-flag method.

Linked lists. Note that the presence-flag solution uses a low-level data structure (Boolean flags) to store the readers of a block, while the B-pointers method saves them in a higher level structure (a fixed array). Perhaps more-flexible data structures, such as linked lists, can be applied to the problem of cache coherence. Distributing the directory updates among multiple processors, rather than a central directory, could reduce memory bottlenecks in large multiprocessor systems.

A few groups have proposed cache-coherence protocols based on a linked list of caches. Adding a cache to (or removing it from) the shared list proceeds in a manner similar to software linked-list modification. Groups using this approach include the IEEE Scalable Coherent Interface (SCI) standard project, a group at the University of Wisconsin-Madison, and Digital Equipment Corporation in its work

with Stanford University's Knowledge Systems Laboratory. The SCI work, which is the most defined of the three, is covered in the report beginning on p. 74. Some features of the Stanford Distributed Directory (SDD) Protocol are outlined on pp. 78-80.

Bus-based schemes. Bus-based systems provide uniform memory access to all processors. This memory organization allows a simpler programming model, making it easier to develop new parallel applications or to move existing applications from a uniprocessor to a parallel system. Since the bus transfers all data within the system, it is the key to performance — and a potential bottleneck — in all bus-based systems.

Several architectural variations of bus-based systems have been proposed. Below, we describe two types — multiple-bus and hierarchical architectures. (See Figure 1.) One of these, the Aquarius multiple-bus multiprocessor architecture, is described in more detail in the report beginning on p. 80.

Multiple-bus systems. An obvious way to extend the single-bus system is to increase the buses. Studies by Hopper et al.[5] show that a system with multiple buses can provide higher bandwidth and performance than a system with wider buses. Multiple buses provide redundancy and extra bandwidth.

A simple extension, splitting a single bus into address and data buses, has a limited performance gain. The next choice is to duplicate buses. This scheme scales the bandwidth linearly with the memory. Synapse[6] had two buses, but they were used for redundancy rather than bandwidth (Figure 1a). (Arbitration and the coherence protocols become complex with multiple buses.)

The Wisconsin Multicube architecture[7] uses a grid of buses connecting the processors to memory. A processor resides at each crosspoint on the grid. The topology allows the system to scale to 1,024 processors. Each processor has a second-level cache that snoops on both the vertical and horizontal buses. (See Figure 1b.)

The Aquarius architecture is based on the same topology as the Wisconsin Multicube. However, it differs in the cache-coherence mechanism and in the distribution of memory modules. The system uses a combination of a snoopy cache-coherence protocol and a directory-based coherence protocol to provide coherency in the system. The memory is distributed per node, unlike the Wisconsin Multicube.

Hierarchical systems. In hierarchical systems, clusters of processors are connected by a bus or an interconnection network. (See Figure 1c.) In the three examples below, the intercluster connection is a bus, and the processors within a cluster are also connected via the bus. This is similar to a single-bus system.

Wilson[8] uses a simulation and an analytical model to examine the performance of a hierarchically connected multiple-bus design. The design explores a uniform memory architecture with global memory at the highest level. It uses hierarchical caches to reduce bus use at various levels and to expand cache-coherency techniques beyond those of a single-bus system. The performance study showed that degradation resulting from cache coherency is minimal for a large system.

The Diffusion Machine[9] is a scalable shared-memory system in which a hierarchy of buses and data controllers link an arbitrary number of processors, each having a large set-associative memory. Each data controller has a set-associative directory containing the state information for its data values. The controller supports remote accesses by snooping on the next bus in the hierarchy in both directions. A cache-coherence protocol enables data migration, duplication, and replacement.

The VMP-MC Multiprocessor[10] is another hierarchically connected multiple-bus multiprocessor system. The first-level cluster comprises processor-cache pairs connected by a bus and is similar to a single-bus system. However, instead of main memory, the bus has an interbus cache module that interfaces to the next bus in the hierarchy. This second-level bus has the main memory. Again, this system provides a uniform memory access. In this system, larger clusters are connected via a ring-based system to provide a large, distributed shared-memory system. ■

Acknowledgment

We would like to thank all the authors of the special reports that follow for their assistance and for their review of this introduction.

References

1. L.M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Trans. Computers*, Vol. C-27, No. 12, Dec. 1978, pp. 1,112-1,118.

2. A. Agarwal et al., "An Evaluation of Directory Schemes for Cache Coherence," *Proc. 15th Int'l Symp. Computer Architecture*, Computer Society Press, Los Alamitos, Calif., Order No. 861 (microfiche only), 1988, pp. 280-289.

3. W.D. Weber and A. Gupta, "Analysis of Cache Invalidation Patterns in Microprocessors," *Proc. ASPLOS III*, Computer Society Press, Los Alamitos, Calif., Order No. 1936, 1989, pp. 243-256.

4. S.J. Eggers and R.H. Katz, "A Characterization of Sharing in Parallel Programs and its Application to Coherence Protocol Evaluation," *Proc. 15th Int'l Symp. Computer Architecture*, Computer Society Press, Los Alamitos, Calif., Order No. 861 (microfiche only), 1988, pp. 373-382.

5. A. Hopper, A. Jones, and D. Lioupis, "Performance Evaluation of Widely Shared Multiprocessors," *Proc. Cache and Interconnect Workshop*, M. Dubois and S. Thakkar, eds., Kluwer Academic Publishers, Norwell, Mass., 1990.

6. S. Frank and A. Inselberg, "Synapse Tightly Coupled Multiprocessor: A New Approach to Solve Old Problems," *Proc. NCC 1984*, AFIPS, Reston, Va., 1984.

7. J.R. Goodman and P. Woest, "The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor," *Proc. 15th Int'l Symp. Computer Architecture*, Computer Society Press, Los Alamitos, Calif., Order No. 861 (microfiche only), 1988, pp. 422-433.

8. A. Wilson, "Hierarchical Cache/Bus Architecture for Shared-Memory Multiprocessors," *Proc. 14th Int'l Symp. Computer Architecture*, Computer Society Press, Los Alamitos, Calif., Order No. 776 (microfiche only), 1987, pp. 422-433.

9. E. Hagersten, S. Haridi, and D.H.D. Warren, "The Cache-Coherence Protocol of the

Data Diffusion Machine," *Proc. Cache and Interconnect Workshop*, M. Dubois and S. Thakkar, eds., Kluwer Academic Publishers, Norwell, Mass., 1990.

10. H.A. Goosen and David R. Cheriton. "Predicting the Performance of Shared Multiprocessor Caches," *Proc. Cache and Interconnect Workshop*, M. Dubois and S. Thakkar, eds, Kluwer Academic Publishers, Norwell, Mass., 1990.

Shreekant Thakkar and **Michel Dubois** are the guest editors of this issue of Computer. Their photographs and biographies appear on p. 11.

Anthony T. Laundrie and **Gurindar S. Sohi** are authors of the report on the Scalable Coherent Interface. Their photographs and biographies appear on p. 77.

Distributed-Directory Scheme:

Scalable Coherent Interface

David V. James, Apple Computer
Anthony T. Laundrie, University of Wisconsin-Madison
Stein Gjessing, University of Oslo
Gurindar S. Sohi, University of Wisconsin-Madison

The Scalable Coherent Interface is a local or extended computer "backplane" interface being defined by an IEEE standard project (P1596). The interconnection is scalable, meaning that up to 64K processor, memory, or I/O nodes can effectively interface to a shared SCI interconnection.

The SCI committee set high-performance design goals of one gigabyte per second per node. As a result, bused backplanes have been replaced by unidirectional point-to-point links. One set of input signals and one set of output signals are defined. Packets are sent to the interconnection through the output link, and packets are returned to the node on the input link.

Although SCI only defines the interface between nodes and the external interconnection, the protocol is being validated on the least expensive and highest performance interconnection topologies, as illustrated in Figure 1.

To support arbitrary interconnections, the committee abandoned the concept of broadcast transactions or eavesdropping third parties. Broadcasts are "nearly impossible" to route efficiently, according to experienced switch designers, and are also hard to make reliable. Because of its large number of nodes and resulting high cumulative error rate, reliability and fault recovery are primary objectives of SCI. Therefore, its cache-coherence protocols are based on directed point-to-point transactions, initiated by a requester (typically the processor) and completed by a responder (typically a memory controller or another processor).

Sharing-list structures. The SCI coherence protocols are based on distributed directories. Each coherently cached block is entered into a list of processors sharing the block. Processors have the option to bypass the coherence protocols for locally cached data, as illustrated in Figure 2.

For every block address, the memory and cache entries have additional tag bits. Part of the memory tag identifies the first processor in the sharing list (called the head); part of each cache tag identifies the previous and following sharing-list entries. For a 64-byte cache block, the tags increase the size of memory and cache entries by four and seven percent, respectively, compared to the traditional eaves-

Project status and information sources

Scalable Coherent Interface. Simulation of the coherence protocols is now under way at the University of Oslo and Dolpin Server Technology in Oslo, Norway. The initial SCI simulation efforts focus on proving the specification's correctness rather than calibrating its performance.

Three University of Oslo researchers (Stein Gjessing, Ellen Munthe-Kaas, and Stein Krogdahl) are formally specifying the intent of the cache-coherence protocol and verifying that the cache updates prescribed by the SCI standard are specified correctly.

The University of Wisconsin's multicube group is now working with the SCI group.

IEEE's SCI-P1596 working group plans to freeze the base coherence protocols by this summer. The group will continue to explore optional coherence extensions to improve the performance of frequently occurring sharing-list updates. If you have interests in this area, please contact the SCI-P1596 working group chair: David B. Gustavson, Computation Research Group, Stanford Linear Accelerator Center, PO Box 4349, Bin 88, Stanford, CA 94309. Gustavson's phone number is (415) 926-2863, his fax number is (415) 961-3530 or 926-3329, and his e-mail address is dbg@slacvm.bitnet.

Stanford Distributed Directory. A group at Stanford University's Knowledge Systems Laboratory is working on simulations to determine the performance of their distributed-directory scheme using linked lists. Further information can be obtained from Manu Thapar, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 701 Welch Road, Palo Alto, CA 94304. Thapar's phone number is (415) 725-3849; his e-mail address is manu@ksl.stanford.edu.

Aquarius. The Aquarius group is evaluating the multi-multi architecture by simulation. Further information on that project can be obtained from Michael Carlton, University of California at Berkeley, Division of Computer Science, 571 Evans Hall, Berkeley, CA 94720. His phone number is (415) 642-8299, and his e-mail address is carlton@ernie.berkeley.edu.