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 As a result, bused backplanes have been replaced by unidirectional point-to-point links.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.

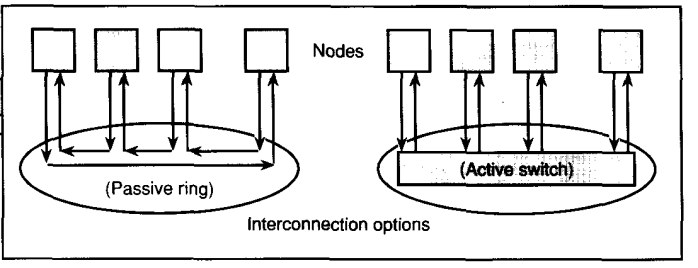

Figure 1. Scalable Coherent Interface (SCI) interconnection models.

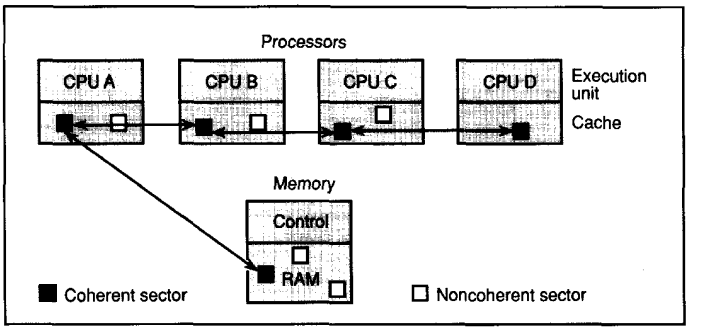

Figure 2. Distributed cache tags.

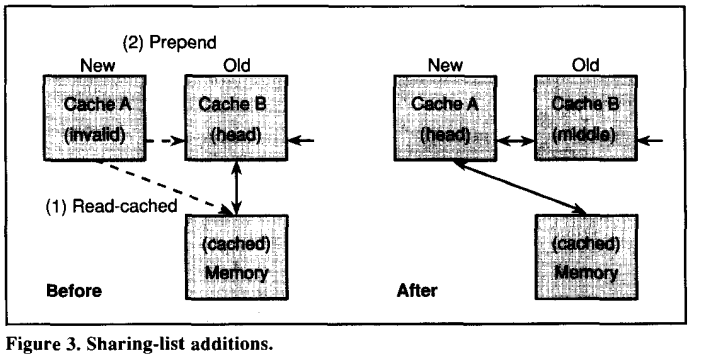

Figure 3. Sharing-list additions.

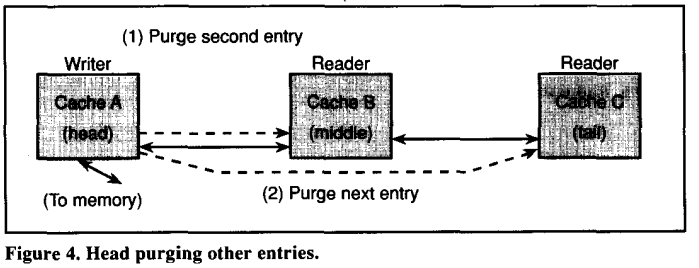

Figure 4. Head purging other entries.

dropping alternatives. However, snoopy protocols have other hidden costs; they require high-performance dual-ported cache-tag memories to allow execution of processor instructions while eavesdropping on other bus activity.

Sharing-list additions. Initially, memory is in the uncached state and cached copies are invalid. A read-cached transaction is directed from the processor to the memory controller. This changes the memory state from uncached to cached and returns the requested data. The data is returned and the requester's cache-entry state is changed from the invalid to the head state.

For subsequent accesses, the memory state is cached, and the head of the sharing list has the (possibly dirty) data. A new requester (Cache A) directs its read-cached transaction to memory, but receives a pointer to Cache B instead of the requested data. A second cache-to-cache transaction, called prepend, is directed from Cache A to Cache B. On receiving the request, Cache B sets its backward pointer to point to Cache A and returns the requested data, as illustrated in Figure 3.

The dotted arrow in Figure 3 illustrates a transaction directed between the processor (the requester) and memory or another processor (the responder). The solid line illustrates the sharing-list pointers. Note that memory cannot always forward the request directly to Cache B — that would create potential deadlocking dependencies.

Unlike the central-directory schemes, request transactions are never blocked at the memory controller; instead, all requests are immediately prepended to the head of the existing sharing list. Requests are added in FIFO order, as defined by the arrival of coherent requests at the memory controller.

Sharing-list removals. The head of the list has the authority to purge other entries to obtain an exclusive (and therefore modifiable) entry. The initial transaction to the second sharing-list entry purges that entry from the sharing list and returns its forward pointer. The forward pointer is used to purge the next (previously the third) sharing-list entry. The process continues until the tail entry is reached, as illustrated in Figure 4. As an option, the purges can be forwarded directly through the sharing-list entries.

Note that purge latencies increase linearly with the number of sharing readers.

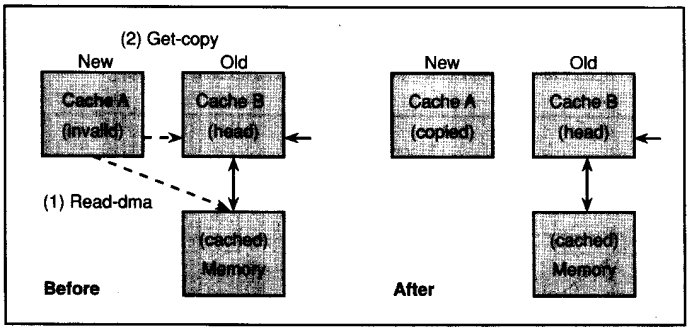

Figure 5. Optimized direct-memory-access reads.

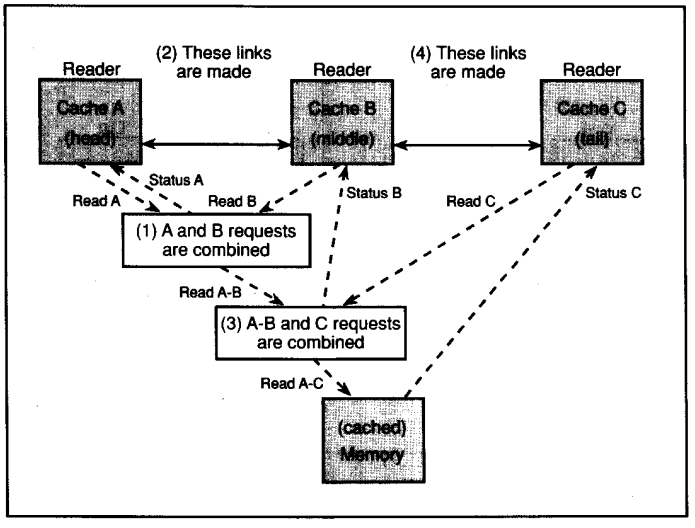

Figure 6. Request combining.

Since purge list sizes are often small, the linear latencies may be acceptable in many configurations.

Entries can also delete themselves from the list when they are needed to cache other block addresses. Since the linked list is distributed and doubly linked, multiple entries can delete themselves simultaneously. Special precedence rules are applied to avoid corruption of pointers when adjacent deletions are initiated simultaneously. To ensure forward progress, the entry closest to the tail has priority and is deleted first.

Standard optimizations. The basic coherence-protocol operations have been optimized to improve the performance of frequent events. We are considering other, more complex optimizations to improve the performance of large system configurations. These compatible extensions to the basic coherence protocols will be included as part of the SCI standard.

An optimized direct-memory-access controller generates read-check transactions to fetch its data from memory. If the addressed location is clean, the data is returned directly from memory; otherwise, the processor is redirected to the current sharing-list head. Thus, the DMA controller can fetch its data without joining the sharing list, as illustrated in Figure 5.

The frequent one-writer/one-reader (producer/consumer) form of data sharing is optimized. The invalidation of the writer (head) and the data fetches of the reader (tail) are both performed as direct cache-to-cache transactions between the head and tail of an established sharing list.

Request combining. One useful feature of linked-list coherence is the possibility of combining list-insertion requests in the interconnection to eliminate hot spots at or near heavily shared memory controllers. Such hot spots degrade performance not only of the requesting processor but also of other transactions that share portions of the congested connection path.

While queued in an active switch buffer, two requests to the same physical memory address (read A and read B) can be combined. The combining generates one response (status A), which is immediately returned to one of the requesters, and one modified request (read A-B), which is routed towards memory. Additional requests (read C) can also be combined with the modified request, as illustrated in Figure 6.

Read transactions and add transactions (add to previous value) can be combined in the interconnection or at the memory controller's front end. Coherent-request combining is simpler than noncoherent fetch-and-add combining,[1] since state need not be saved in the interconnection while the modified request is being forwarded to memory.

SCI's optional extensions. The latency of distributing data or purges to large numbers of readers currently scales linearly with the number of read-sharing processors. We are investigating the use of

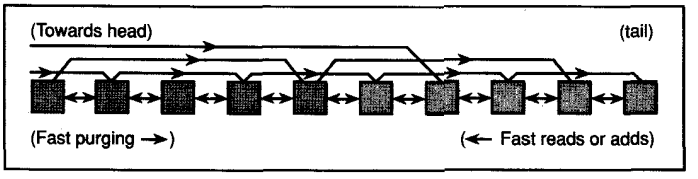

Figure 7. Redundant sharing-list pointers.

redundant pointers to reduce these linear delays to logarithmic latencies (order log(N), where N is the number of read-sharing processors). The redundant pointers can be created while the request combining is being performed, to provide the binary tree-like structure illustrated in Figure 7.

The redundant pointers could be used by multiple readers, to request early copies of heavily shared data, or by a writer, to quickly purge stale copies when a new data value is written.

Synchronization. In shared-memory architectures, locks are the primary form of synchronization for large-scale multiprocessors and must be handled efficiently. The SCI options include efficient synchronization primitives for large-scale multiprocessors. A queued-on-lock-bit idea, described by Goodman, Vernon, and Woest,[2] provides FIFO access to synchronization variables. Since linked cache entries form a queue, little additional hardware is needed to implement an SCI variant of this scheme. The advantage of the queued-lock scheme is that (except for replacements) lock requests are serviced in FIFO order and only $O(N)$ transactions are generated. ■

Acknowledgments

The IEEE P1596 Scalable Coherent Interface project was started as a study group, under the name of SuperBus, by Paul Sweazey. Dave Gustavson is now the chair and is responsible for the continuing development efforts. Others initially or currently involved with the cache-coherence issues include Knut Alnes, Jim Goodman, Marit Jensen, Ernst Kristiansen, Stein Krogdahl, John Moussouris, Ellen Munthe-Kaas, Alan Smith, and Hans Wiggers.

We would like to acknowledge recent contributions from Jim Goodman and others at the University of Wisconsin that have simplified the basic proposals and triggered many of the SCI project's continuing investigations.

References

1. G.F. Pfister et al., "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proc. Int'l Conf. Parallel Processing*, Computer Society Press, Los Alamitos, Calif., Order No. 637 (microfiche only), 1985, pp. 764-771.

2. J.R. Goodman, M.K. Vernon, and P.J. Woest, "Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessors," *Proc. ASPLOS III*, Computer Society Press, Los Alamitos, Calif., Order No. 1936, 1989, pp. 64-75.

David V. James, a research scientist at Apple Computer, is a major participant in several IEEE bus standards. He is the chair of the IEEE P1212 CSR Architecture working group as well as a member of the directly affected bus standards (IEEE P896.2-Futurebus+, P1596-SCI, and P1394-Serialbus). His research interest is scalable interconnection architectures, from low-cost mouse interfaces to high-performance massively parallel processors.

James holds BS and MS degrees in electrical engineering and a PhD degree in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a member of IEEE, ACM, Eta Kappa Nu, and Tau Beta Pi.

Anthony T. Laundrie received the BSEE and MSEE degrees in 1987 and 1990 from the University of Wisconsin-Madison. The past four years have been spent writing design automation software and integrating memory hardware for the Astronautics ZS-1 minisupercomputer.

Now back in school at the University of Wisconsin-Madison, he is studying memory architectures for high-performance systems in greater detail. Laundrie is a member of IEEE, Eta Kappa Nu, Tau Beta Pi, and Phi Kappa Phi.

Stein Gjessing is an associate professor at the University of Oslo, Norway. He is presently head of the Department of Informatics. His research interests are in concurrent programming, operating systems, and formal specification and verification of computer programs. His work on SCI is supported by the Norwegian Research Council NTNF.

Gjessing received his PhD in computer science from the University of Oslo in 1985.

Gurindar S. Sohi has been with the Computer Sciences Department at the University of Wisconsin-Madison since September 1985. He is currently an assistant professor. His interests are in computer architecture, parallel and distributed processing, and fault-tolerant computing.

Sohi received his BE degree, with honors, in electrical engineering from the Birla Institute of Science and Technology, Pilani, India, in 1981 and the MS and PhD degrees in electrical engineering from the University of Illinois, Urbana-Champaign, in 1983 and 1985.