

ABSTRACT

There is rapidly increasing demand for very-high-performance networked communication for workstation clusters, distributed databases, multiprocessors, industrial data acquisition and control systems, shared access to distributed data, and so on. Higher-bandwidth hardware using the traditional protocols is not sufficient. Even at 100 Mb/s, and certainly at 250 Mb/s, throughput for many applications is so limited by delays due to architecturally induced inefficiencies, such as software overheads (often hundreds of microseconds), that higher bandwidth generally raises cost without improving performance. A new approach to communication is required, one that can eliminate the delay due to software overheads, if we are to reap the full benefit of the far higher bandwidths that modern hardware can provide. SCI solves this problem by using the distributed-shared-memory paradigm, typically offering submicrosecond delays and bandwidths currently in the range of 1250 to 8000 Mb/s per network node. This article first reviews the general properties that an appropriate system architecture should have, and introduces an architectural model, the Local Area MultiProcessor, distinguished by its shared-memory performance and its ability to handle LAN-style distances. These desired properties are then considered in more detail, and practical design decisions are made, illustrated by the evolution of the ISO/ANSI/IEEE standard Scalable Coherent Interface (SCI) as it addressed these issues. Finally, the current status of the various SCI follow-on and support projects is reported.

The Scalable Coherent Interface (SCI)

David B. Gustavson and Qiang Li, Santa Clara University

The technology described in this article arose from a project whose goal was a standard interconnect that would enable the use of many mass-produced microprocessors for supercomputer-class computation at low cost. The usefulness of this technology for local area network (LAN) communication was an unexpected consequence of the scalability requirements of that project, so the technology is described in language appropriate for high-speed buses or memory systems as well as for LANs. This work is closely related to the IEEE Std 1394 Serial Bus, and to the IEEE P1394.2 Serial Express project, which has just started. This technology is contrasted with Serial Bus and Serial Express later in this article.

The supercomputing goal demands communication among multiple processors at the speed of main memory systems, with submicrosecond latency and multigigabyte-per-second bandwidths. There is no time for software-implemented protocols in such a system, so the communication model must be kept extremely simple: load from and store to memory that is in a single flat address space, but physically distributed among the processors. The physical-layer signaling mechanisms also must be kept very simple so that the interface circuits can run at very high speed without becoming expensive.

However, this memory-like model is very general, so once the physical links proved capable of LAN distances the distributed-memory model became capable of LAN applications, backward-compatible with traditional protocols as needed during the transition to this new technology, yet fundamentally more efficient, able to utilize large numbers of gigabyte-per-second communication links effectively.

The use of the distributed memory model for networking is not new [1, 2], but these schemes had neither the very high bandwidths nor the support for caching provided by the Scalable Coherent Interface (SCI) technology.

In the following, we return to the original motivation for developing the SCI (local area multiprocessor); its application to networking will be discussed in the fourth section.

To meet the ever-increasing demand for computing power and the ever-decreasing expectation for its cost, the use of highly parallel multiprocessor systems is essential. Only a few of the traditional few-processor supercomputer companies have survived, and they are now adding multiprocessor-based product lines.

It has been obvious for a long time that the cost effectiveness of the microprocessor is unexcelled, and microprocessor performance continues to grow exponentially. The difficulty has been learning how to solve hard problems by using many inexpensive processors instead of one extremely powerful (and expensive) processor.

The Scalable Coherent Interface is an interconnect technology designed to be scalable and cost-effective. The objectives of the interconnection can be summarized as follows: We seek a scalable abstract specification of the interface to communicating components that are approximately at the level of microprocessors (intelligent I/O devices, memory, bridges to other systems, networks, etc.) so that these components can be assembled into more complex and capable multiprocessor systems. We require support for efficient multiprocessing using both the message-passing and distributed-shared-memory models. Because multiple cached copies of data will be present in most systems, we require this interface to include a cache coherence mechanism to keep these copies consistent.

HISTORY AND BACKGROUND

The Scalable Coherent Interface was developed by a number of high-performance bus designers and system architects who had come to understand the fundamental limits to bus technology during their work on Fastbus (IEEE 960) and Futurebus+ (IEEE 896.x). These contemporary buses pushed bus signaling technology to its limits, and provided various architectural features that support the use of multiple processors.

However, it was recognized that very soon microprocessor speeds would exceed the capability of any bus to support significant multiprocessing, and our efforts of the preceding decade were fated to a short life. A study group was organized in 1987 to look for some way out of this catastrophe. This Superbus Study Group met from November 1987 through July 1988, examining alternative approaches for providing bus-like services while avoiding bus limitations. Gradually the form that the solution would require, if indeed a solution were possible, became clear. In July 1988 an official IEEE Working Group was chartered. The group was very fortunate to have several very talented and experienced core members who were available essentially full-time. Initially all the work-

Buses have intrinsic limitations due to signaling technology: bus transmission lines are imperfect because they have taps where the various devices connect, which form stubs that cause reflections.

ing group members had a bus-oriented point of view, so the group started by pushing against the bus limits and gradually evolved to a broader perspective.

Buses have intrinsic limitations due to signaling technology: bus transmission lines are imperfect because they have taps where the various devices connect, which form stubs that cause reflections. Bus drivers require too much current and switch it very rapidly, causing system noise. Wire-OR functions, commonly used on certain bus signals, have incorrect values for durations proportional to the physical length of the bus, unless the bus is driven by current sources (rarely the case — all commercial drivers are low output impedance drivers, i.e., voltage sources). Bus transceivers and connectors load the transmission lines, and are far from ideal. But the most fundamental limitation is that the bus is a shared resource that can be used by only one transmitter at a time: a bus is inherently a bottleneck.

Therefore, buses cannot support a large number of processors, especially not fast ones. While their useful life can be extended a bit by cleverness and brute force, the potential gains are relatively small and the costs become very high. For example, doubling the width of a bus does not double its speed because there are fixed overheads associated with arbitration (deciding who gets to use it next) and addressing. Lengthening data transfers to reduce the effect of these overheads is of little use once the data blocks exceed the size of the most common transfers, usually cache lines.

Signaling speeds can be increased by shortening the bus, but that makes it less useful. Reducing the signal voltage helps, but the practical reduction is limited by noise sensitivity problems. Using multiple buses to perform more than one transfer at a time results in a complex (expensive) bus-bridge mechanism to maintain cache consistency (coherence) in shared-memory systems that use bus-snooping technology.

In retrospect, many of the solutions are obvious: what matters is how much useful bandwidth is delivered, which is a function of the transmission speed and the efficiency of the protocol used. The design must balance often conflicting requirements while maximizing its utility. A protocol that uses the bandwidth very efficiently, but as a result reduces the speed of transmission because of its complexity, is not providing the best value for the user. RISC principles are cost-effective in protocols as well as in instruction sets.

We discuss in two groups the solutions chosen by SCI designers: those maximizing the physical layer speed, and those supporting high-performance system design.

MAXIMIZING PHYSICAL-LAYER SPEED

- Improve the transmission lines by eliminating the multiple connectors and taps, allowing only one transmitter and one receiver. This allows the signaling speed to be increased significantly. The bus becomes a point-to-point link.
- Reduce the voltage swing of the signals to reduce the required driver currents and to make it easier to drive the signals at higher frequencies.
- Use differential signaling to reduce noise sensitivity in the receivers, especially important when using smaller signal voltages, and to eliminate the generation of high-speed ground-current fluctuations (system noise) by fast driver circuits.

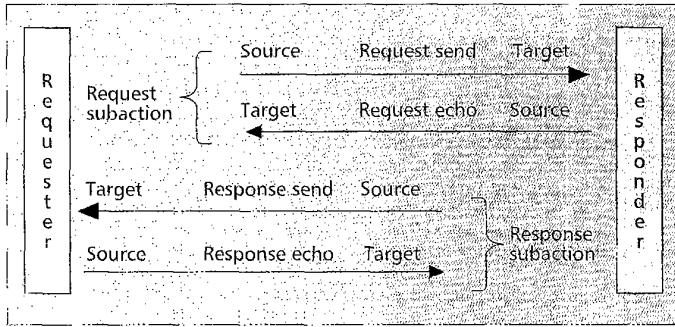
- Avoid using reverse-propagating flow control (a wait-a-moment signal). Reverse flow control causes a scaling problem, where the receiver must anticipate running out of buffer space by an amount equal to at least twice the amount of data in flight in the transmission line, so any given interface implementation sets a bound on the total cable or fiber length of the link. Reverse flow control also complicates signal amplification and retiming, which we will

discuss later. When one considers bit-serial fiber links, it becomes clear that reverse flow control must be multiplexed with the data which is flowing in the opposite direction on another link, and then it becomes obvious that flow control should be part of the normal data stream. Thus, the flow of data on the outgoing link must be controlled in some way by information received on the incoming link. This control may include prohibition or permission to send more data, acknowledgments, or requests for retransmission.

- Unidirectional links and the need for flow control imply that each node in the system needs at least one input link and one output link. For a general-use interface, these links should be identical. For maximum versatility, the protocol allows these pairs of links to be connected in several ways, representing a very wide range of cost/performance trade-offs, ranging from single rings to meshes of rings connected by trivial switches (two nodes back to back with expanded address decoding capability), to small switches connecting several rings, to large switches connecting many single nodes.

SUPPORTING HIGH-PERFORMANCE LARGE SYSTEM DESIGN

- The protocol does not require in-order delivery, as that places difficult constraints on switches and is incompatible with the use of alternate routing for enhanced reliability or improved performance. However, the system must be able to ensure in-order delivery when that is needed. (This can be achieved by waiting for end-to-end confirmation of delivery before sending the next item, accepting a resulting performance penalty.)
- Unidirectional links imply that the protocols must be based on separate requests and responses, so the system is inherently split-transaction. (Transactions are also split on high-performance buses so that other traffic can use the bus bandwidth during the interval between the request and the response.)
- A request or response is a packet containing all the information needed for its delivery (the full destination address and return address), the operation to be performed upon delivery, and, if appropriate, also the data.
- To minimize latency, the packet header is kept small; the routing address is placed at the front; the error-detecting code is placed at the tail end of the packet rather than in the header (so the header can be transmitted before the tail arrives). This means that cut-through routing [3] can be used, where a packet starts out the other side of a bridge or switch before it has been entirely received or



■ Figure 1. Transaction phases.

checked. This greatly reduces latency in the case of a moderately loaded system.

- Caches are essential to most mechanisms that reduce the effective cost of accessing remote data. Caches introduce the problem of inconsistent copies of data, which must be solved efficiently.

These and similar logical considerations led to the SCI design being completed in an unexpectedly short time. The work proceeded by consensus: if the right choice (of the detailed design features needed to implement our objectives) is obvious, take it. If not, make an arbitrary choice and follow it until problems are found. In many cases the working group was so pleased to have finally found *one* solution which met the requirements that consensus was more nearly *relief* — it was not obvious that good solutions even existed. Only one vote was taken by the working group, a final (unanimous) vote to declare the project complete. The SCI specification was essentially completed by January 1991, then underwent final editing and polishing before gaining final approval by the IEEE Standards Board in March 1992, and by the American National Standards Institute (ANSI) in October 1992. Official printing was delayed until August 1993 due to various schedule conflicts, but the final draft was technically solid and readily available, and was used for interface chip design.

ANSI/IEEE STANDARD 1596, SCALABLE COHERENT INTERFACE

A standard SCI module defines signals, connector, and power for operation at a link speed of 1000 Mbytes/s. A standard cable connector defines the use of these signals for applications where the module form factor is not appropriate, over short distances (meters). A standard fiber optic serial interface solves interface problems over longer distances (kilometers) at a link speed of 1250 Mb/s. The same bit-stream may be sent over coaxial cable at lower cost for medium distances (tens of meters). Other speeds and signaling technologies will be standardized in the future as appropriate.

SOLVING THE BUS SIGNALING AND BOTTLENECK PROBLEMS

The propagation velocity, and thus the latency, of a packet is always limited by the speed of light. Fortunately, computer scientists know techniques (e.g., caching and prefetching) that can compensate for the bad effects of latency.

There are also techniques that can compensate for low bandwidth, such as compression and caching, but these are not sufficient for all situations. Fortunately, bandwidth is becoming less and less expensive, and has no inherent limits. (Links can be paralleled to reach

any desired bandwidth, but there is no analogous way to reduce the latency.)

SCI needed two fundamental changes from the way a bus transmits information. First, to make signaling speed independent of the size of the system, interfaces do not wait for each signal to propagate (i.e., a bus cycle) before sending the next. This means each communication is performed by sending packets. Packets include an address, command, and data as needed.

The packet is either accepted and stored in the responder's queues or discarded (if there is not sufficient space); an echo (acknowledgment) packet tells the requester whether it can discard its send packet or

must retransmit it later because it was not accepted. A similar handshake occurs on the response, as shown in Fig. 1.

SCI send packets (Fig. 2) have a 16-byte header that contains address, command, transaction identifier, timestamp, and (in a response) status information. All packets that may need storage space in queues are multiples of 16 bytes, to simplify storage management at very high speeds. The echo packet is an 8-byte subset of the header (not shown here); it is never stored in queues. Echoes provide a ring-local handshake that passes responsibility for a packet from a transmit queue to the next receive queue as the packet flows through the interconnect. Thus, the echo provides flow control, not a confirmation of the completion of the subaction.

Requests are confirmed end-to-end by responses for reads, writes, and locks. Moves are unconfirmed writes, more efficient when reliable transfer is not critical (e.g., writes to a video refresh buffer). Events are not only unconfirmed by responses, they also have no flow control (echo). Their initial intended use was to send a time mark for precise time-of-day synchronization. Some applications may use events to transfer data, but these must ensure that there is always sufficient buffer space to receive the data, or be such that lost data is not critical. Typically, this requires special hardware designed for the specific application. A few transactions need an extended header (not shown), which adds another 16 bytes immediately following the header. The error detection code is a 16-bit cyclic redundancy check (CRC) at the end of the packet. For space-accounting purposes, it is counted as part of the 16 bytes of the header.

The second major change compared to a bus is that SCI uses multiple signal paths (links), so multiple independent transfers can take place concurrently. For high performance one can use separate links for each processor, memory, or I/O device.

The SCI links are fast and narrow because interface pins are always relatively expensive. In order to reach high speeds,

	Request subaction		Response subaction	
Readxx*	Header		Header	0,16,64,256
Writexx*	Header	16,64,256	Header	
Movexx*	Header	0,16,64,256		
Eventxx*	Header	0,16,64,256		
Locksb	Header	16	Header	16

Note: xx represents one of the allowed data block lengths (number of data bytes on the right after the header)

Adapted from IEEE std 1596-1992

■ Figure 2. Transaction formats.

low-voltage differential signals are used. SCI initially used 16-bit-wide ECL-compatible signals because of the industry experience with and support for that standard. Future links will use even lower voltages (IEEE 1596.3-1996, Low Voltage Differential Signals, LVDS) that are chosen for compatibility with very large scale integrated (VLSI) complementary metal oxide semiconductor (CMOS), BiCMOS, and GaAs circuitry.

The links run continuously, sending idle symbols when no packets are being transmitted, so the receiver can remain perfectly synchronized at all times, ready for action. This reduces latency: SCI packets do not need the packet prolog that is essential for synchronizing receivers in bused networks (e.g., Ethernet).

In a high-performance SCI system, a vendor-dependent switch accepts packets from nodes and routes them to other nodes as specified by the address in the packet. SCI does not specify the internal details of such a switch, because many interoperable cost/performance trade-offs are possible.

Lowest-cost SCI systems, such as desktop systems, typically use a ring connection instead of a switch, connecting one node's output link to its neighbor's input link. The interface circuit was made more complex by the decision to support rings, because a node may receive a packet intended for some other node and have to pass it along. That requires some buffer memory and some address recognition logic. However, the result is that a single interface definition works over a very wide range of applications, increasing the production volume and lowering the costs for everyone.

The design of an SCI node (Fig. 3) is largely constrained by the considerations stated previously. Data arriving on the IN link have to be resynchronized to the node's own clock. This is done by the receiver's elastic buffer circuitry. The rest of the node circuitry is synchronous, which greatly simplifies the design of the ultrafast first-in first-out buffers (FIFOs) and logic.

If a node receives a packet intended for some other node while transmitting a packet of its own, part or all of the incoming packet is stored in the bypass FIFO until the output link becomes available again.

When no packet is being received, idle symbols keep the receiver synchronized and carry information about the priorities of other nodes. They also carry go-bits, which act rather like tokens and help ensure fair use of the links. Fair use of part of the bandwidth is important for avoiding starvation or deadlock.

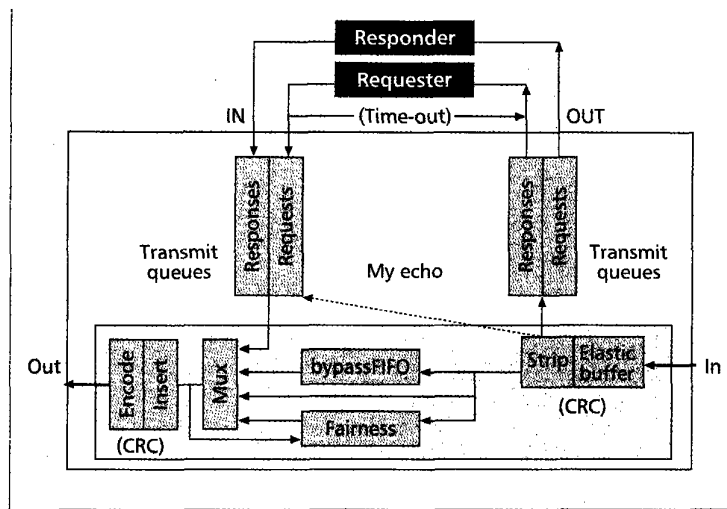
An SCI node maintains dual queues to keep responses independent of requests. Without dual queues, it would be possible for excessive requests to prevent the sending of responses, resulting in deadlock.

SCI defines efficient packet-based protocols that provide the kinds of services one expects from a computer bus. The main differences seen by the user are related to the separation of the request for service from the response.

More-sophisticated buses split the operation into a request and a response phase, just as SCI does. Then the interface has to keep track of pending requests in order to match the responses to them, and a different style of mutual exclusion becomes necessary.

SOLVING THE CACHE COHERENCE PROBLEM

Cache consistency, or coherence, has been maintained in small bused systems by taking advantage of the bus



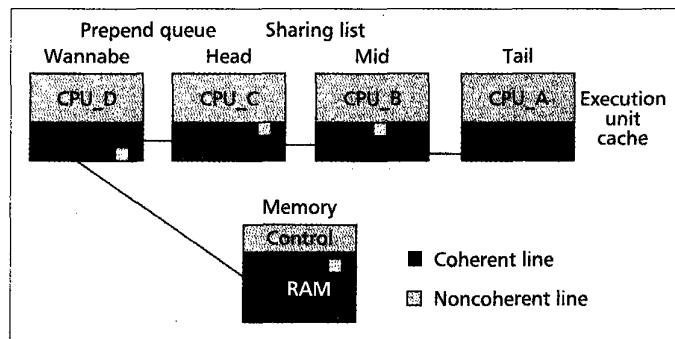
■ Figure 3. Node interface structure.

bottleneck: every cache controller observes every transaction in the system, "snooping" to catch transactions that might invalidate cached data [4, 5]. That approach can be scaled up to a few buses by making the bridges rather sophisticated, but it does not work in highly parallel systems.

A cache coherence scheme that scales to large systems requires a directory to keep track of which data are being used by which caches so that the appropriate caches can be updated as necessary. Earlier schemes have used a directory but kept it in memory. SCI maintains it as a distributed doubly linked list of caches instead, with the head pointer at a memory controller and the link pointers stored in the cache controllers (Fig. 4). This has the virtue that the correct amount of storage is always available for the directory structure, no matter how many caches are sharing copies of a particular line of data. It also spreads the maintenance traffic across the system, rather than concentrating it at the memory. Even though these linked-list structures are shared, they are designed to be updated concurrently by multiple processors without any semaphores or lock variables. (The protocols in effect use indivisible compare-and-swap transactions instead.)

The coherence mechanism operates as follows: the cache/memory controller of any processor that has a cache miss issues an SCI request to the memory that was addressed. When that request reaches the memory, the memory controller unconditionally swaps the requester's 16-bit nodeId (the pointer to the requester) with the saved nodeId of the previous requester (the pointer to the previous requester) for those data.

That swap gives each requester a pointer to its predecessor.



■ Figure 4. Distributed cache tags.

A four-processor node is now being built by several manufacturers, taking advantage of the native multiprocessor bus features of modern microprocessors such as the Pentium Pro.

sor, forming a singly linked list, the *prepend queue*, which can be built up at the full speed of the memory controller. If the memory has valid data, it returns the data along with the pointer; if not, it returns only the pointer and a status indication. In any case, the requester contacts the previous requester, if any, and asks to become doubly linked.

When it has become doubly linked (and has received the data from the predecessor if not from the memory) it also becomes the *head* of the doubly linked *sharing list*, a unique member of that list. It can keep the headship until it is asked to link to its successor (and consents). Only the head is permitted to modify data (and only if its original request notified memory of this intention). If it does so, it also has the duty to follow the doubly linked list to invalidate all the other copies, eliminating the sharing list.

Any cache may leave the sharing list at any time (e.g., for cache-line rollout), and a sharing reader that later wishes to modify the data must first roll out from the list and then request the data from memory again, with permission to modify, joining the prepend queue of "wannabe" nodes and proceeding as above until it becomes head. This mechanism precisely defines the order of changes, ensures exclusivity for the writer, and also has useful forward-progress properties.

The cache coherence problem can be avoided if memory is not shared. This approach is used in multicomputers, which rely on message passing (explicit interprocessor communication) and explicit information management by software.

If an application does not need cache coherence, the coherence features of SCI can be ignored. The existence of coherence in the SCI architecture costs very little if it is not used, approximately two bits in the command code field of the packet header.

If coherence is implemented but data happen not to be shared, the existence of coherence in the SCI architecture causes essentially no performance penalty; that is, no extra transactions will be generated for linking directories or other coherence maintenance.

SIGNALING CONSIDERATIONS

There are several reasons for not using reverse flow control in SCI. In addition to the buffer scaling issues discussed previously, reverse flow control also complicates the signal amplification and retiming that may be needed in long links, requires an extra set of synchronization logic in the transmitter, and approximately doubles the cost of bit-serial fiber links. Reverse flow control implies that the timing of the reversed signal at the transmitter, relative to the transmitter clock, depends on the length of the link. This opens the undesirable possibility of exposing synchronizer errors in the driver at certain cable lengths and not at others, which would significantly complicate the diagnosis of problems in a large system.

The link runs continuously and in one direction only, so differential-driver current remains constant, whereas reversing a link would require first turning the driver off, which would drop the signal and ground current to zero, generating noise; turning the other end's driver on would reverse the ground current, generating noise again. This also implies differential termination at the receiver instead of series termination at the driver, which would save power but would cause short pulses in the signal, ground, and power-supply currents that would

have a duration of double the transmission line delay, generating system noise.

Each node uses its own clock oscillator, because distributing a central clock is expensive and scales inconveniently. The transmitter's clock should be included with the data on each link to simplify the receiver's task of correctly sampling the incoming data. Nodes can use the incoming link clock as their own in some applications, but in general this introduces system hazards due

to jitter and instability (due to cascaded stages of phase-locked loops), so nodes should be capable of resynchronizing the incoming data to their own clock. Clocks can be assumed to have nearly the same frequency, within the tolerances of inexpensive crystals. Such systems are sometimes called *pleiosynchronous*.

This implies that the protocol must provide sufficient opportunities for inserting or deleting nondata harmlessly in an elastic receive buffer that resynchronizes the incoming data to the local clock domain.

A related problem is handling skew on parallel links, caused by cabling imperfections that deliver some bits before or after others. It is desirable for the protocol to provide a way for the receiver to observe skew, so it can adjust the sampling delays individually to compensate. This allows the use of inexpensive cables.

THE TOPOLOGY ISSUES

The topology of an SCI network is very flexible. There may be multiple devices on a single SCI node (e.g., an entire multiprocessor with its I/O subsystems and memory). There may be multiple SCI nodes paralleled on a single device, for example, multiple SCI interfaces and links simply used in parallel where extremely high bandwidth or high reliability is needed [6]. Organizing the links as pairs of counterrotating rings [7] is also attractive for both performance and reliability; or a single device may use multiple SCI links orthogonally, for moving data across the various dimensions of a multidimensional mesh.

All SCI interfaces support daisy-chaining their links to form rings that can stand alone (no hub or switch is required) or be interconnected by switches. The number of devices daisy-chained can be chosen independently at each switch port to vary the cost/performance trade-off as desired.

A four-processor node is now being built by several manufacturers, taking advantage of the native multiprocessor bus features of modern microprocessors such as the Pentium Pro, which uses SCI to assemble these into larger multiprocessors.

Simple 2 x 2 switches [8] can be used to connect multiple rings to form larger systems, or to split rings into smaller rings to increase their effective bandwidth. The interconnect topology can be chosen to match the communication traffic patterns of the algorithms and applications being supported, whether regular patterns such as 2-D meshes or Omega networks, or ad hoc patterns typical of data acquisition systems. Switches with more ports may be able to offer higher performance, particularly for random access patterns. The internal switch mechanism may use arbitrary technology for a wide variety of cost/performance tradeoffs, as long as the external ports obey the SCI interface protocol. Generally, this requires each switch port to have a buffer that can hold part or all of an SCI packet, as needed, during contention.

MULTIPROCESSOR ISSUES

In a cache-coherent environment, mutual exclusion can be handled by the mechanism that guarantees exclusive use of a cache line by a writer. The processor can temporarily lock an exclusive cache line while it does any operations it desires, then release it to other readers or writers. However, coherence may not be available in all cases. For example, when accessing (through a bridge) a bus that does not support cache coherence, one may need to tell the bridge to do a read-modify-write.

Therefore, SCI includes a set of lock protocols that experience shows to be useful for a variety of purposes: masked swap, compare and swap, and fetch and add [9, 10]. Each of these primitives sends the command and necessary data to a destination device (perhaps a bridge or memory controller), which performs the operation indivisibly, then returns the result to the requester. This mechanism works well through switches or other interconnects. (It even works well on buses.) (Related work has shown how to handle synchronized updates of multiple items with minimal hardware support, of interest for distributed databases [11-13].)

It is desirable for user code to include lock instructions, and for the processor interface to execute these in cache or via remote transactions, depending on whether the address is cacheable. However, processors do not yet directly support this.

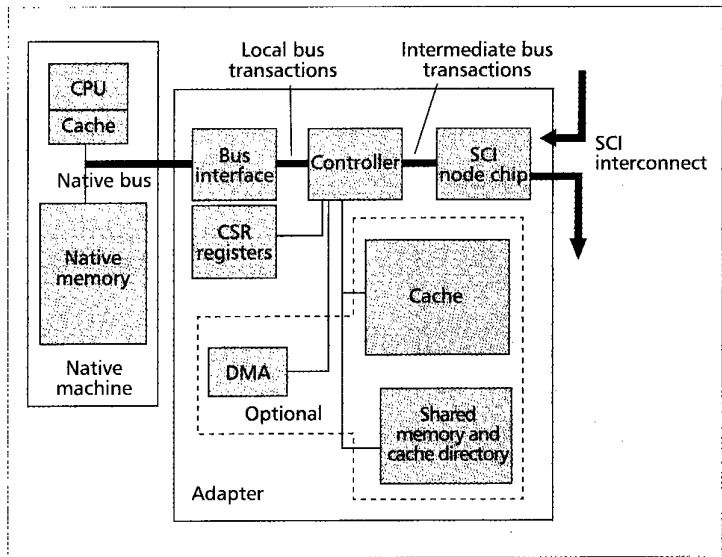
A particularly interesting use of swap operations is in the maintenance of shared lists that hold information for an interrupt-servicing processor or commands for a direct memory access (DMA) controller, for example. If these lists are properly structured, multiple processors can add items to them while one processor takes items off, without any need for semaphores or system calls.

For example, a clean way to handle interrupts is to associate a particular shared list with one priority of interrupt service and a specific bit in an interrupt-triggering control register. To request interrupt service, add a work item to the appropriate list describing what needs to be done, then set the list's bit in the interrupt register. The item in this list is similar to the interrupt vector in single-processor systems. When the processor is ready to service that priority of interrupts, it takes work items off the list and services them.

This mechanism is clean because all storage allocation is done by the service requester, so there is no danger of the server running out of storage when the work piles up (a possible cause of deadlock). The interrupt bit is simple to implement because it is only a latch, with no FIFO storage or critical timing implied. Setting the bit merely alerts the processor that the list should be checked; the processor can clear the bit as soon as it commits to look at the list.

A common use of mutual exclusion is to grant sequential use of one resource to a series of requesters. This process can generate a large amount of useless interconnect traffic as processors keep updating their cached copies of the shared exclusion variable. SCI defines a queue on lock bit (QOLB) mechanism that uses the linked lists of the cache-coherence system to pass the resource efficiently from one processor to the next.

Statistics on memory sharing are controversial, because there are few relevant machines in existence, and the usage patterns on any real machine will evolve to optimize performance on that architecture. However, most researchers agree that unshared data is the most common case, followed by pairwise shared, followed by multiply shared. Thus, the pairwise sharing case may be an important one to optimize. SCI



■ Figure 5. Adapter design.

defines optional pairwise sharing optimizations that allow two processors to pass data back and forth without interacting with memory, thus distributing system traffic and reducing activity at the memory controllers.

APPLICATIONS

The essence of SCI is well illuminated by an architectural model called the local area multiprocessor (LAMP). LAMP is a hybrid of the network of workstations (NOW) and massively parallel processor (MPP). LAMP is a distributed-shared-memory multiprocessor. A LAMP consists of a number of units, where each can be a shared-memory multiprocessor or workstation. The units can be distributed over a local area (to tens of kilometers). LAMP is a nonuniform memory architecture (NUMA) or cache-coherent NUMA (CC-NUMA) machine. From a user's point of view, it is similar to an MPP system. LAMP has all the advantages of NOW (low cost, incremental upgrades, multiple justifications for purchase, etc.), while significantly reducing NOW's long latency problem. This architectural model has been widely accepted by industry. However, it is clear that a new interconnect technology is needed that can meet a difficult requirement: providing very high bandwidth and very low latency, while allowing connections over a local area.

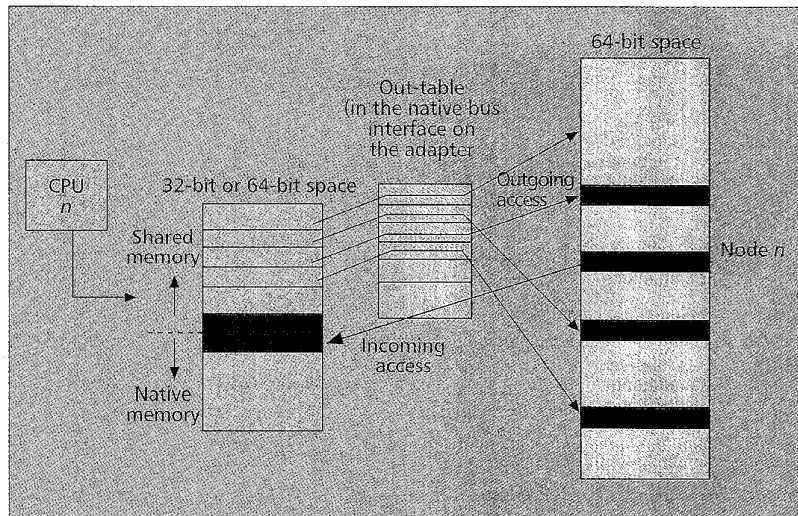
The first commercial machine to use SCI in essentially this way was the Convex Exemplar [6], which differs from a LAMP mainly in packaging and intent. That is, all the "workstations" are inside the Convex box, and no external user connections to the SCI links are allowed.

There are also non-SCI-based machines that support the distributed shared-memory model. Examples of these include Alewife [10], FLASH [14], SHRIMP [15], Butterfly (noncoherent) [16], and KSR [17]. These machines are either research prototypes or use proprietary interconnection networks. None of these attempt LAN-style out-of-box distances.

We will use a hypothetical design to illustrate how SCI can be used to connect workstations to form a LAMP. Note that the discussion below is in no way exhaustive.

AN ARCHITECTURAL MODEL

Figure 5 shows an adapter plugged into an existing machine. All components in the existing machine are prefixed with "native," and all components on the adapter are referred to as "local." Depending on the existing machine, the native bus



■ Figure 6. Map local physical memory space to the 64-bit global space.

can be a processor bus (such as the Pentium Pro [18] processor bus, Sun MBus [19], or Macintosh PowerPC bus [20]) or an I/O bus (such as PCI). The shared memory, cache, and DMA on the adapter are optional.

The SCI node chip is a VLSI chip that handles all SCI transactions and translates between SCI and the intermediate bus. Second-generation node chips (both CMOS and GaAs) are already available [8, 18, 21, 22]. The intermediate bus is supported by the node chip as an interface to various applications. The bus interface is a machine-dependent component; its function is to provide an interface between the native bus and the adapter. If the native bus uses a 64-bit address, the bus interface is mostly passing-through wires. If the native bus uses a 32-bit address, a mapping table is needed (discussed later). The local cache controller interprets transactions from the native bus and the remote machines via the intermediate bus and generates control sequences to access the local cache or memory.

Figure 6 illustrates the relationship between a processor's physical memory space and the SCI global memory space. When the native bus uses 32-bit addresses, part of the 64-bit global space is mapped into the local physical space. The "out-table" is part of the bus interface and can be configured dynamically by the operating system. If the native bus uses 64-bit addresses, such a map is not necessary, but it provides useful flexibility.

When the central processing unit (CPU) issues a memory access, a corresponding physical address X will appear on the native bus. Depending on the value of X , the memory access will be performed to the native memory, local memory, or remote memory. In the case of remote memory, an SCI transaction will be sent through the interconnection network to the appropriate node. Though not shown in the figure, mapping the SCI addresses to the native addresses through an in-table adds useful flexibility and protection.

The shared memory, cache, and DMA on the adapter are optional. DMA is very desirable for medium to large data transfers between machines. The difference between this DMA and a typical DMA in the native machine is that this one can be designed to be accessible by remote processors, while the native one may be kept "off limits" by the design of the native architecture. This DMA can be designed to work in the user's space for efficient data transfer without operating system (OS) overhead [4]. If one does not want the native memory exported, the memory on the adapter can serve as shared memory. Furthermore, when the cache of the adapter

is present, the shared memory will exhibit cache-only memory architecture (COMA)-like [23, 24] behavior, which is very desirable for parallel programming because it reduces the programmer's burden for optimizing data processor allocation.

OPERATING SYSTEMS FOR LAMP

LAMP is architecturally a NUMA or CC-NUMA machine, nothing new. The difference is that NUMA or CC-NUMA traditionally implied a tightly coupled machine, while LAMP may include a blend of tight and loose coupling, depending on how its components are distributed and how it is used. At one extreme, a large number of machines connected by SCI might be stacked in racks or hidden in cabinets and locked up in a machine room, resulting in a normal tightly coupled, shared-memory machine; we call this the LAMP-1

model. At the other extreme, a number of machines may be spread out across a building or a campus, each perhaps independently owned and operated, but connected by SCI. In this case, the LAMP becomes a "network of machines" with a shared-memory interconnect; we call this the LAMP-3 model. There is also a middle ground, where machines are distributed physically but managed and operated centrally; that is, the end users cannot become superusers, reboot the system, mount the file system, change priorities, or shut down machines in the system. We call this the LAMP-2 model. In principle, arbitrary mixtures of these models can be interconnected and can coexist, though management of such systems is difficult and will require some experience and evolution. Heterogeneous systems will eventually be the norm, but with additional software complexity that will be avoided in initial implementations.

The decision for selecting an operating system is mainly a trade-off between tightly coupled sharing and fault tolerance. The operating system for LAMP-1 is like that of any NUMA or CC-NUMA machine. Resources can be fully shared, including kernel data structures. For LAMP-3, however, sharing needs to be carefully tailored: memory can be shared in user space subject to OS protection; remote I/O operations must be started by the remote OS; file system buffer cache may not be directly shared; and so on. In this case, the OS running on each individual machine is similar to a single-processor or SMP operating system, depending on what the individual machine is, except it needs to manage remote memory accesses.

ESTABLISHING SHARED MEMORY IN THE USER'S VIRTUAL SPACE

Once two machines have each other's memory mapped into SCI's physical-address space, shared memory in virtual memory can be established by mapping the physical memory pages into the user's virtual space. For example, UNIX System V IPC can be used to establish shared memory [25]. Note that many applications would choose to share only certain regions of memory, and efficient applications will actively share data as little as possible to avoid the inherent costs of remote accesses.

MESSAGE PASSING IN SHARED MEMORY

For any communication mechanism for computer systems, the ultimate goal is to bring data into a computational context, that is, into data structures which can be accessed by the proces-

Fiber ribbon offers a very attractive match to SCI, and several demonstration projects have shown operation of ribbon cables at the full SCI gigabyte-per-second link speed.

sor. For communication mechanisms based on I/O models, data will need to be packaged by software to fit the I/O semantics, and unpacked to bring data back into the computational context. Furthermore, I/O-model communication implies multiple logical channels multiplexed onto a few I/O ports, which requires OS intervention for protection and fairness on a message-by-message basis; that adds a significant amount of overhead. Memory-based communication mechanisms such as SCI, however, bypass the computation-I/O semantic translation. Hardware maps a practically unlimited number of logical channels (any memory cell to any remote memory cell) onto the physical media without OS intervention. The protection is done when setting up the memory mapping, a much less frequent operation than handling it at message-passing time. By reducing or eliminating the software overhead, intermachine communication latency can be brought down to the microsecond or even submicrosecond level.

We examine two cases of message passing: solicited messages (receiving process polling for messages) and unsolicited messages (receiving process sleeping).

In parallel processing, often a number of collaborating processes start together and establish communication at starting time. The subsequent message passing is done in planned patterns; these are solicited messages. In this case, the receivers actively check for incoming messages. On LAMP, the "channels" between processes are simply memory buffers in user space, so message passing does not need OS intervention except for establishing the mapping of the shared buffer into the process's space. There is no execution-time overhead incurred in this case.

In the more general case, messages are unsolicited. That is, the receivers do not know when messages are coming or how many messages are coming. In this case, a sender needs to interrupt the receiver's OS so that the receiver can be awakened after a message is put into its message buffer. The receiver will not be able to process the message until it eventually gets context-switched in. This interrupt is necessary for any I/O-based communication, and the context switch time incurred is the result of multitasking on the receiving machine, not the inherent overhead of the message-passing mechanism. Any message-passing mechanism in this case will have the context switch overhead unless the receiving process is polling for messages, in which case they are solicited messages.

Where compatibility with popular network protocols, such as Transmission Control Protocol/Internet Protocol (TCP/IP), is required, the traditional protocols can be implemented on top of the shared memory. However, the software protocol translation will dissipate much of the performance advantage gained by using shared memory. Some studies of TCP/IP implementations on shared memory have shown similar performance to TCP/IP on Ethernet [26]. This will be the case wherever the protocols dominate the link speed in limiting performance.

SHARED I/O SYSTEM

Sharing memory among workstations can change the access to remote I/O systems. One can DMA data blocks from a local disk to remote memory directly, making the I/O systems shared to a large extent. In a conservative design in which DMA can only be programmed by the local OS, one can per-

form a remote disk access by passing the local memory buffer address and the data address on the remote disk to the OS controlling that disk, and the OS can program its DMA to carry out the request. A more aggressive approach is to allow DMA to be programmed by a remote processor. In that case, it is possible to have remote disk access without remote OS intervention. In contrast, accessing a remote disk on a message-passing system will require a significant amount of

overhead to buffer the data at the disk's site, decompose the data into packets by software, and send them to the requesting site.

Besides enabling efficient remote disk access, shared memory also makes the implementation of distributed file systems more interesting. Buffer cache can now be shared among machines, which provides a large memory pool for buffer cache and allows the implementation of efficient buffer cache management schemes. For fault-tolerant applications, convenient access to remote memory and I/O systems facilitates checkpointing and logging without specialized equipment. Also, software implementation of redundant arrays of inexpensive disks (RAID) across workstations becomes natural.

A distributed DMA architecture suitable for such systems has been defined in IEEE project P1285, Standard for Scalable Storage Interface.

RELATED STANDARDS PROJECTS AND FUTURE PLANS

There are several projects [27] extending or supporting SCI, including P1596.1, Guide to SCI Bridges; P1596.2, Cache Optimizations for Large Number of SCI Processors (Kiloprocessor Extensions); 1596.3, Low Voltage Differential Signals for SCI (standard approved in March 1996, defines new signals and 4-, 8-, 32-, 64-, and 128-bit-wide links); 1596.4, High Bandwidth Memory Chip Interface, RamLink (standard approved in March 1996); 1596.5, Shared Data Formats Optimized for SCI (standard approved in 1993); P1596.6, SCI/RT, Real-time Extensions for SCI; P1596.7, Standard for A High-Speed Memory Interface (SyncLink, a bus-based version of RamLink); P1596.8, Standard for Parallel Links for the Scalable Coherent Interface standardizing cables, connectors, and fiber ribbon links for this and the next-generation SCI; P1596.9, Standard for Physical Layer Application Programming Interface for the Scalable Coherent Interface (SCI API).

As technology advances, SCI will define new link standards to carry the same packets at still higher bandwidths. Fiber ribbon offers a very attractive match to SCI, and several demonstration projects have shown operation of ribbon cables at the full SCI gigabyte-per-second link speed. One recent commercial product, Motorola's Optobus™, was apparently designed expressly for SCI. Optical connections eliminate many potential problems caused by electrical interference or ground-potential differences between SCI nodes. Though multimode fiber optic links are limited to much shorter distances than single-mode, they are also much less expensive.

Similarly, there are enormous application possibilities for slower links. An eight-bit-wide link operating at 250 Mbytes/s or 500 Mbytes/s might be a good match for the next generation of PCs. Perhaps it will be appropriate soon to define a PC form factor and signal standard for SCI, based on new

CMOS chips or processors with integrated SCI interfaces.

It is becoming apparent that the same RISC-like simplicity that enables SCI to run at such high speeds makes it inexpensive to implement at low speeds, and with its narrow links and completely integrated interface (transceivers and all), SCI will become the least expensive "bus" as soon as production volumes are sufficient.

While designing SCI, the working group learned a lot about how the processor should interact with the interconnect, keeping in mind what is possible for an interconnect to do and what is not. The group is considering how best to spread this information to processor designers, to make multiprocessor systems more efficient. Possibly this could be a recommended practice, or even a standardized clean 64-bit RISC architecture optimized for use with SCI.

Please refer to the SCI standard for more information about SCI. This standard was intended to be read by humans, with details mostly relegated to machine-readable C code (included on a diskette inside the back cover). There are a number of companies using SCI, or derivatives of it, for developing new products [6, 7, 18, 21, 22, 28-31].

SCI AND OTHER COMMUNICATION MECHANISMS

The main difference between SCI and other communication mechanisms is that SCI's design aims at the system architecture as a whole, not merely at moving data.

Given today's technology, there are many ways to achieve gigabit-per-second bandwidth with hardware latency on the order of tens of microseconds. Although SCI can achieve higher bandwidth and lower latency than the other commonly known standards, the raw performance of its lowest-speed, bit-serial, links (1.25 Gb/s) is similar to the others. However, the bit transmission rate is of no direct relevance to the user; what really matters is the time to send useful amounts of data from an application on one machine to another application on a different machine (i.e., overall system performance, including software overhead, affects the user). For example, asynchronous transfer mode's (ATM's) latency, largely caused by software overhead, can easily be 10 times higher than the hardware transmission time [32]. This is not the fault of any particular mechanism; rather, it is a common phenomenon in computer communication systems that is likely a consequence of the historical evolution of I/O systems.

SCI, originally designed for multiprocessor systems, has its communication mechanism in a critical path as part of the memory system. Therefore, SCI protocols had to be designed for minimal latency and for essentially zero software overhead. When applied to the LAN environment, this has the very desirable effect that moving data from one application to another does not require software intervention, especially for shared-memory parallel programming on multiple workstations. Thus, users who really need high performance can handle the communication within their own applications, with minimal software overhead. Of course, when TCP/IP protocols are used on top of SCI hardware for backward compatibility reasons, the software overhead will again dominate the performance [26]. However, as the use of shared-memory-based network protocols evolves, the protocol overhead

It seems likely that competition will eventually push prices down to normal market levels, reducing SCI's costs below those of other bus interfaces, which are typically far more complex.

should be reduced significantly.

Theoretically, any communication mechanism that carries data from one machine to another can be used to implement shared memory with appropriate hardware logic on the interface. The question is how efficiently it can be done and the performance one can achieve. SCI's advantage is that it has been optimized for shared memory, with all the critical components built in, such as the mapping to address space, synchronization, cache

coherence, and so on.

In general, SCI's capabilities overlap all others whose distances are limited to LAN territory (i.e., tens of kilometers or less), so the decision whether to use SCI or some other technology in traditional LAN or I/O-channel applications must be based on economics, backward compatibility considerations, the value of low latency in the given application, and so forth.

SCI becomes inefficient over longer distances, unable to keep a link fully occupied because its protocols restrict any single node to no more than 64 subaction packets outstanding at one time. One could work around this by using multiple nodes at each end of a long link, and there might be applications where one wishes to use SCI over a long link just for protocol consistency despite its inefficiency, but in most cases bridging SCI to a wide area service like ATM would be preferred.

SCI protocols presume that the links are reliable, which improves the efficiency of normal operation. The validity of this assumption is monitored by CRC and time-outs. Normal delays, such as waiting for space to become available in a queue, do not trigger time-outs, but lost or corrupted packets trigger time-outs very rapidly. Recovery from such errors is handled by software, and making the recovery process efficient was not a goal of the SCI design. If links are unreliable, they must be either replaced or repaired to make them reliable, or enhanced by a layer of protocols using redundant information and error correction technology in a way that is transparent to the SCI protocols. SCI is robust, supporting guaranteed delivery and controllable ordering while handling contention and flow control transient conditions, but depends on reliable links. The usual methods of substituting spare links or routing around a defect can be applied to SCI, but the standard does not specify how to do this.

SCI was designed for ease of interfacing to other protocols, including essential transactions such as single-byte writes and recognizable lock transactions, without which the interface problem is much harder. Thus, there will be bridges between SCI and VME, PCI, ATM, FibreChannel, Serial Bus, and so on. In the long run, for equal production volumes, interfacing SCI directly to I/O devices should be the least expensive solution; however, earlier technologies may become so inexpensive due to their high production volumes that they persist indefinitely.

The biggest disadvantage of SCI at present is the high cost and scarcity of interface components. This is due to the market's initial perception that SCI's high performance limits it to low-volume markets, so components are priced to amortize their development over a small production quantity. In reality, however, SCI's interface chips are dominated by the cost of their internal high-speed memory, 500 to 1000 bytes, with relatively simple logic compared to today's common bus interfaces. Thus, it seems likely that competition will eventually push prices down to normal market levels, reducing SCI's costs below those of other bus interfaces, which are typically far more complex.

FIBRECHANNEL

FibreChannel attempts to incorporate all that has been learned by the industry's leading experts on I/O channels and networks. It is a rich architecture that provides many attractive features. However, the channel and network approach does not scale upward to the performance levels needed for the next generation of machines.

The latency for communication with FibreChannel has been reported as exceeding 1000 ms, and while great improvements will undoubtedly be made, there is no likelihood of reaching the $\leq 10 \mu\text{s}$ latencies needed for fine-grained parallelism. FibreChannel performs best when transferring long blocks of data, and will find effective applications where that is a dominant consideration.

FibreChannel is adding support for ring connections, the "arbitrated loop," and there are likely to be peripherals such as disk drives built with that interface. Switches exist, though typical latencies are reported to exceed $10 \mu\text{s}/\text{stage}$. Interoperability has been a problem because various interfaces were built to different drafts of the evolving standard.

FibreChannel is often considered to be a gigabit-per-second technology; indeed, chips exist which can operate at that speed. However, the marketplace centers on the quarter-speed version because throughput in practice is completely dominated by overheads, so the higher costs of the full-speed links do not result in any significant increase in throughput. It appears that FibreChannel at birth has already reached its performance scaling limit.

ATM

ATM is destined to become the high bandwidth long distance data connection service. Its characteristics are well matched to its original design goal of transmitting digitized voice, but less well matched to other data. In particular, a key part of the ATM switch strategy allows discarding occasional packets under peak loading conditions. This hardly causes any degradation for a voice connection, but seriously impacts computer data transmission since one small omission may cause a considerable delay for retransmission. Bandwidth reservations and flow control mechanisms will gradually bring these problems under control.

Since the high-volume wide-area telephony market has not yet developed for ATM, manufacturers of ATM components have tried to sell into the multiprocessor interconnect and network market. This has met with only modest success, since the peculiarities of ATM significantly impact its performance in this area.

For example, ATM switches have been observed to have latencies exceeding $10 \mu\text{s}$. The ATM protocols do not imply such large latencies, but $10 \mu\text{s}$ is considered to be insignificant in the telephone industry. The computer industry has not offered high enough volume to have much influence on the switch design.

In terms of bandwidth, SCI offers 1 Gbyte/s (parallel) or 1

CACHE COHERENCE

To reduce the effective memory-access delay, blocks of data are transferred from main (relatively slow) or remote (very slow) memory into a high-speed cache memory that has a very fast connection to a processor, for subsequent fast access to adjacent data or repeated access to the same data. In a multiprocessor system, there can be copies of the same block of data in many caches at the same time. If one processor modifies the data, all the other copies become obsolete and incorrect. To eliminate this problem, those copies are either discarded or updated by *cache coherence* protocols performed by hardware or software. A goal of SCI was to maintain coherence by hardware so that caches may be used to improve performance without modifying application software (i.e., they are transparent to the application).

NUMA

Nonuniform memory architecture. Although desirable, providing uniform memory access delay from all processors to all memory in a large multiprocessor system is very difficult, often only feasible by making all memory uniformly slow. NUMA represents a compromise that allows each processor to have fast access to its local memory and slower access to all other memory.

CC-NUMA

Cache-coherent NUMA reduces the impact of slow remote memory access by caching remote memory data near each processor as needed. The caches may also significantly reduce the interconnect traffic.

COMA

COMA stands for cache-only memory architecture. In a traditional memory architecture, the memory address of a data item gives the unique location of that data in the physical memory. In a COMA machine, a data item does not have a fixed location. Data is moved dynamically to be close to the processor that accesses the data most. The address is merely a search key for its current location in physical memory. The task of a programmer is changed from optimizing data allocation in a traditional memory architecture to maximizing locality of computation, which may be much easier.

LAMP

LAMP stands for local area multiprocessor. It refers to systems that are logically similar to traditional tightly coupled multiprocessors, such as shared memory and shared I/O systems, but can be physically distributed over a local area, such as a building or campus. Depending on the actual implementation, a LAMP can be a NUMA, CC-NUMA, or COMA machine.

NOW

Network of workstations, a project of the University of California, Berkeley. The goal is to use a network of workstations with low interprocessor communication latency for cooperative computing.

Gb/s (serial). Today, typical ATM link speeds are 155 Mb/s, and the major push is toward lower speeds, such as 25 Mb/s.

SCI and ATM address different needs. SCI supports shared memory to reduce interprocessor communication latency and software overhead. It also provides cache coherence for effective parallel processing. If one considers SCI as specifying two layers of services, the shared memory layer and the communication layer, ATM specifies only the communication layer. It is not ATM's goal to address the issues of software latency in a parallel computing environment. Thus, SCI is suitable for high-performance parallel and distributed computing over relatively short distances (kilometers), while ATM is more suited for wide-area communication.

It is possible, in principle, to use ATM as the communication layer for SCI, particularly where long distances are needed or when one does not need the maximum performance of SCI. SCI packets could be transported transparently over ATM. The 48-byte payload size of ATM packets is not a good

*One can use the high
bandwidth of SCI to
transport ATM packets
among a number of ATM
ports, forming an ATM
switch. Such a switch could
have the advantage of being
user-configurable and
expandable.*

match to typical cache lines of 32 or 64 bytes, but this can be handled by the interface. The main technical difficulty for such a design is translating the different semantics of error handling and flow control between SCI and ATM. Because of these differences, all designers to date have decided to treat the interface between SCI and ATM essentially as an interface between memory and ATM, so the interface moves specified data to/from SCI "memory" over an ATM link, the traditional I/O model. There have been several projects designing such SCI-to-ATM interfaces, but none are commercially available yet.

Using SCI to implement ATM switches, on the other hand, is attractive, and such designs are underway. One can use the high bandwidth of SCI to transport ATM packets among a number of ATM ports, forming an ATM switch. Such a switch could have the advantage of being user-configurable and expandable. Some optimizations to SCI are desirable for handling 48-byte packets efficiently, and are being incorporated in the next-generation standard.

SERIAL BUS

Serial Bus, IEEE Standard 1394-1995, also known as FireWire™, is particularly compatible with SCI, sharing many architectural features, especially the distributed-memory model and memory-addressing bus-like protocol. Serial Bus was designed as a distributed I/O bus, emphasizing low cost, ease of use, and ruggedness. It does not attempt to deal with the multiprocessor issues, concurrent transactions, longer distances, higher speeds, or cache coherence that SCI was designed to handle.

Serial Bus operates at 100 or 200 Mb/s, with a potential for reaching 400 Mb/s. Its cables are very thin and flexible, containing only three pairs of conductors (one pair carries power that can be used to power small devices or to keep interface circuits active even when their attached devices are powered off). Cable length limits and timing considerations limit the span of a single Serial Bus to approximately a large room. To interconnect an entire house one must bridge several Serial Buses.

Serial Bus devices include digital recording camcorders, already on the market, and various computer peripherals (including hard disk drives) are being prepared for market by several manufacturers.

SERIAL EXPRESS, P1394.2

Serial Bus has received considerable consumer-market-volume adoption, generating an urgent need for an expansion path to higher performance and longer distances, such as one needs for fully interconnecting a modern home or office building for video, audio, control, and so on.

As a result, a new strategy is being implemented that will combine SCI and Serial Bus to form a next-generation protocol which includes the best features of both; that is, the scalability, concurrency, multiprocessor support, and long links of SCI, and the isochronous transport of Serial Bus, plus the real-time arbitration model developed for SCI/RT, plus a 32-bit CRC, plus local tags in the packet header (to make interface chips even simpler than SCI's).

This combined protocol is called Serial Express (courtesy of Intel, which has abandoned its own plans for gigabit links in favor of joining this effort). The design has been moving

very rapidly, pushed by an intensive collaboration of Sun and Apple, joined by Intel and many other companies as it moved into the open standards arena officially on April 8, 1996, releasing its first public draft on the same day it began the formal IEEE standardization process at the Microprocessor Standards Committee as IEEE P1394.2.

The basic protocol and the serial links will be developed by the Serial Bus community, with the aid of SCI's leading architects, and the SCI community will develop the higher-performance parallel links and cache coherence protocols

needed for multiprocessor applications. Products should start appearing approximately 1997-1998.

OTHERS

There are many other standard interconnects in use that have some overlap with SCI. These include Ethernet at 10 and now 100 Mb/s, fiber distributed data interface (FDDI), high-performance parallel interface (HIPPI) and now SuperHIPPI, and others. All suffer from communication latencies that seriously limit their performance in fine-grained parallel processing. Some proprietary methods for reducing the latency problem are becoming known, but none yet support cache coherence in a distributed system [2].

REFERENCES

- [1] G. Delp *et al.*, "Memory as a Network Abstraction," *IEEE Network*, July 1991, pp. 34-41.
- [2] R. Gillett, "Memory Channel Network for PCI," *IEEE Micro*, Feb. 1996, pp. 12-18.
- [3] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Communication Switching Technique," *Computer Networks*, vol. 3, no. 4, 1979, pp. 267-86.
- [4] J. Archibald and J. Baer, "Cache Coherence Protocols: Evaluation Using Multiprocessors," *ACM Trans. on Comp. Sys.*, vol. 4, no. 4, Nov. 1986, pp. 273-98.
- [5] J. R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. 10th Annual Int'l. Symp. on Comp. Architecture*, 1983, pp. 124-31.
- [6] CONVEX Computer Corporation, "Exemplar Architecture Manual," Richardson, TX, 1993.
- [7] S. Scott, "The GigaRing Channel," *IEEE Micro*, Feb. 1996, pp. 27-34.
- [8] Dolphin Interconnect Solutions, Inc., "A Backside Link (B-Link) for Scalable Coherent Interface (SCI) Nodes," 1994, Westlake, CA.
- [9] D. V. James and G. D. Stone, "Synchronization Primitives: Scaling to Distributed Systems," *Proc. 1st Int'l. Workshop on SCI-based High-Performance Low-Cost Comp. (SCIZZL-1)*, Santa Clara Univ., Aug. 1994, pp. 1-16.
- [10] A. Agarwal *et al.*, "The MIT Alewife Machine: A Large Scale Distributed-Memory Multiprocessor," *Proc. Workshop on Scalable Shared Memory Multiprocessors*, June 1991.
- [11] D. V. James and D. Singer, "Nonblocking Shared-Data Updates Using Conditional Lock Primitives," *Proc. 2nd Int'l. Workshop on SCI-based High-Performance Low-Cost Computing (SCIZZL-2)*, Santa Clara Univ., Mar. 1995, pp. 67-74.
- [12] J. M. Stone *et al.*, "Multiple Reservations and the Oklahoma Update," *IEEE Parallel & Distrib. Tech.*, Nov. 1993, pp. 58-71.
- [13] M. Herlihy and J. E. B. Moss, "Transactional Memory: Architectural Support for Lock-Free Data Structures," *Proc. 20th Int'l. Symp. Comp. Architecture*, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 289-300.
- [14] J. Kuskin *et al.*, "The Stanford FLASH Multiprocessor," *Proc. 21st Int'l. Symp. on Comp. Architecture*, Apr. 1994.
- [15] M. A. Blumrich *et al.*, "A Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *Proc. 21st Int'l. Symp. on Comp. Architecture*, Apr. 1994, pp. 142-53.
- [16] W. Crowther *et al.*, "The Butterfly (TM) Parallel Processor," *IEEE Comp. Architecture Technical Committee Newsletter*, Sept. 1985, pp. 18-45.
- [17] J. Rothie, "Overview of the KSR1 Computer System," Kendall Square Research Report TR9202001.
- [18] Dolphin Interconnect Solutions, Inc., "P6-SCI Board Hardware Specifi-

- cation," 1996, Westlake, CA.
- [19] Sun Microsystems, "SPARC MBus Interface Specification," Apr. 1991.
- [20] IBM and Motorola, "PowerPC 601 RISC Processor User's Manual."
- [21] Dolphin Interconnect Solutions, Inc., "PCI-SCI Bridge Functional Specification," Westlake, CA, 1996.
- [22] Vitesse Semiconductor, "VSC 7201a SCI DataPump Datasheet," Camarillo, CA.
- [23] E. Hagersten, A. Landin, and S. Haridi, "DDM-A Cache-Only Memory Architecture," *Computer*, vol. 25, no. 9, Sept. 1992, pp. 44-54.
- [24] P. Stenstrom, T. Joe, and A. Gupta, "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architecture," *Proc. 19th Annual Int'l. Symp. on Computer Architecture*, 1992.
- [25] M. Bach, *The Design of the UNIX Operating System*, Upper Saddle River, NJ: Prentice Hall, 1986.
- [26] J. Gorman and Q. Li, "A Study on TCP/IP Performance on Shared Memory Platforms," *Proc. 5th Int'l. Workshop on SCI-based High-Performance Low-Cost Computing (SClzzL-5)*, Santa Clara Univ., Mar. 1996, pp. 21-28.
- [27] SClzzL and SCI-related standards information, contact information, and so on available at <http://www.SClzzL.com>.
- [28] D. Engebretsen et al., "Parallel Fiber-Optic SCI Links," *IEEE Micro*, Feb. 1996, pp. 20-26.
- [29] W. Nation, "Aspects of Full-Speed SCI-Link Silicon Implementations," *Proc. 1st Int'l. Workshop on SCI-Based High-Performance Low-Cost Computing (SClzzL-1)*, Santa Clara Univ., Aug. 17-18, 1994, pp. 50-57.
- [30] R. Safranek, "Considerations in Implementing a System Based on SCI," *Proc. 4th Int'l. Workshop on SCI-based High-Performance/Low-Cost Computing (SClzzL-4)*, Crete, Greece, Oct. 1995, pp. 12-22.
- [31] Vitesse Semiconductor, "VSC 7203 2-port Switch Datasheet," Camarillo, CA.

The design has been moving very rapidly, pushed by an intensive collaboration of Sun and Apple, joined by Intel and many other companies as it moved into the open standards arena officially on April 8, 1996.

- [32] J. Hennessey and D. Patterson, *Computer Architecture-a Quantitative Approach*, 2nd Ed., Chapter 8, Morgan Kaufman, 1996.

ADDITIONAL READING

- [1] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*, 2nd Ed., Benjamin/Cummings, 1994, pp. 577-80.
- [2] M. A. Blumrich et al., "Protected, User-level DMA for the SHRIMP Network Interface," *Proc. 2nd Int'l. Symp. on High Performance Architecture*, San Jose, CA, Feb. 3-7, 1996, pp. 154-65.

BIOGRAPHIES

DAVID B. GUSTAVSON earned his Ph.D. in experimental high energy physics from Stanford University. He is currently the executive director of SClzzL and a research professor in the Department of Computer Engineering at Santa Clara University. Gustavson is the chairman of the IEEE Computer Society's Microprocessor Standards Committee, served as chair of the Scalable Coherent Interface working group, and is active in numerous standards projects in the area of high-performance multiprocessor interconnect architecture.

QIANG LI earned his B.S. in electrical engineering from Xi'an Jiaotong University, Xi'an, China, and M.S. and Ph.D. in computer science from Florida International University. He is currently an associate professor in the Department of Computer Engineering at Santa Clara University. Qiang Li's research interests include parallel and distributed computing, particularly in the area of operating system support for distributed shared physical memory, parallel I/O, and fault tolerance.