The Integrated Cluster Bus for the IBM S/390 Parallel Sysplex

by T. A. Gregg
K. M. Pandey
R. K. Errickson

IBM has developed a new S/390® Parallel Sysplex® coupling interface for the G5 server called the Integrated Cluster Bus (ICB). This interface improves the coupling efficiency by greatly reducing message-passing latency. Using the transport layer of the S/390 self-timed interface (STI) introduced in the G3 server, ICB adds channel function to the hub chip to allow a more direct interconnection between S/390 servers. This new channel has the same function as the present intersystem channel (ISC), but because it is integrated into the hub chip and therefore requires no additional components, its reliability is much better than that of the ISC. Since the ISC transmits data at a peak rate of 106 MB/s over distances exceeding ten kilometers and the ICB transmits data at a peak rate of 333 MB/s at distances of ten meters, the ISC is still required for the more geographically dispersed Parallel Sysplexes, whereas the ICB is well suited to the machine room, where multiple servers can be interconnected by ten-meter cables. This paper describes the design approach for the ICB. It describes the fundamental message-passing requirements of the Parallel Sysplex and how they are implemented in very complex yet compact hardware in the server's hub chip.

Introduction

The S/390* Parallel Sysplex* requires high-speed interconnections to pass messages between instances of the operating system, OS/390*, and the coupling facility (CF), one or more of which is physically located in an S/390 server [1, 2]. The interconnection used in all earlier generations of S/390 servers, bipolar and CMOS, is the intersystem channel (ISC). This interface is optical and bit-serial, operating at 100 MB/s at distances of up to ten kilometers. When introduced in 1994, the ISC was perfectly suited to the performance and distance requirements existing at the time. To meet the low-latency requirements of message passing, the short messages did not require extremely high bandwidth. Instead, the ISC design concentrated on a low-overhead protocol, and kept microcode path lengths short by putting as much function as practical into hardware state machines. Relatively long distances were required to interconnect the large bipolar servers together with their input and output devices, of the order of at least 100 meters. The same interface can also be used by the more geographically dispersed computer clusters.

As the CMOS servers replaced bipolar, new environments required new solutions. First, the CMOS servers are physically much smaller than the bipolar servers. At present, several (up to eight) servers can be interconnected by relatively short (in the range of ten meters) cables. Second, new workloads move more data and still require low latency, thus requiring higher

0018-8646/99/$5.00 © 1999 IBM

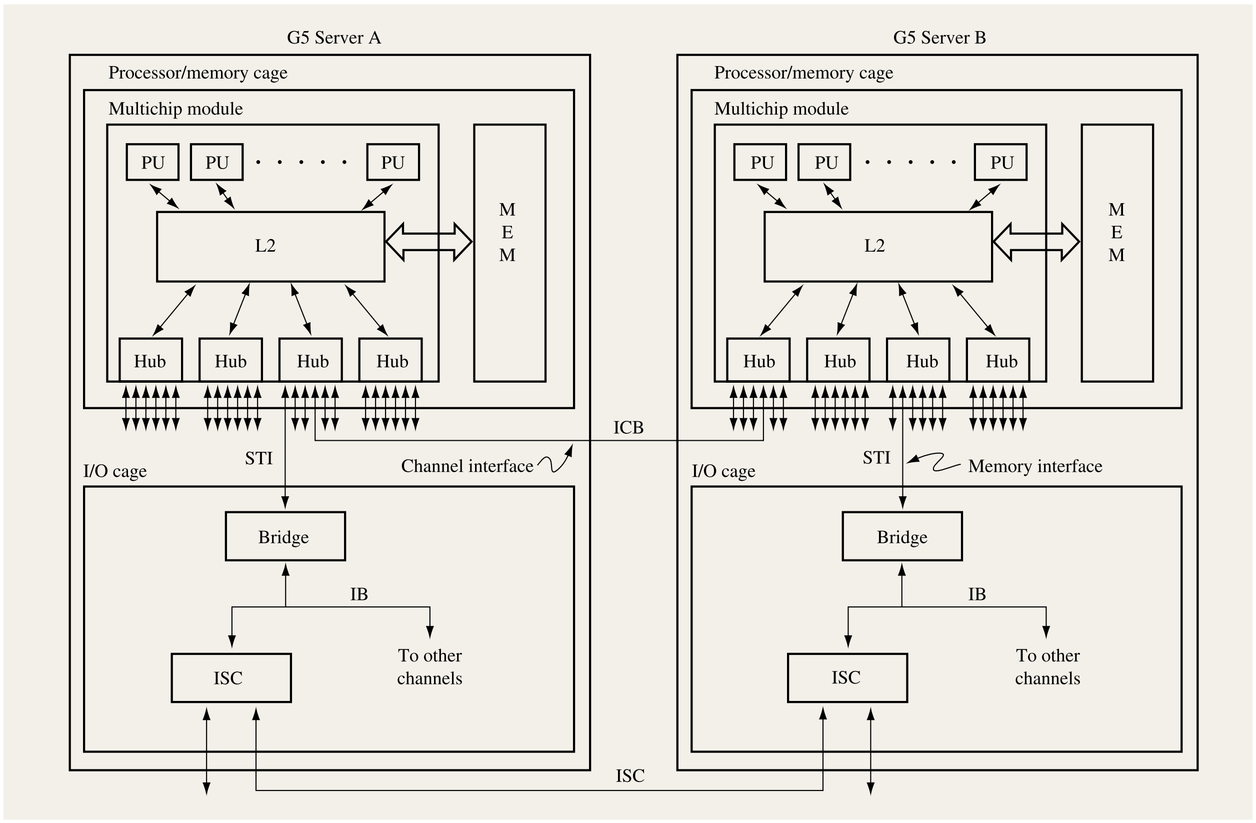

Figure 1

Interconnection of two G5 servers.

bandwidths. Third, S/390 Parallel Sysplex uses synchronous message passing: The processor stops executing new instructions until the current message has been completed by the coupling facility. This implies that the message-passing implementation must scale with the processor speeds to keep the percentage of processor wait time low [3, 4]. ISC hardware, on the other hand, is often retained by the customer when upgrading processors from one generation to the next, so their relative performance is reduced, and coupling efficiency is consequently reduced. Finally, to both improve the overall server hardware failure rate and lower the cost, a solution using less hardware than ISC was required.

Since the smaller size of the CMOS servers allows most of the connections to be accommodated by relatively short cables, we considered using the S/390 self-timed interface (STI) technology as a new interconnect. Using this technology required more than simply plugging an STI port on one server into an STI port on another server. The channel function of the ISC had to be optimized in a way that permitted it to be moved into the hub. Since each of the six STI ports on the hub chip requires a channel, the channel function also had to be compact.

Hardware structure of ISC and ICB

Figure 1 shows how two G5 servers are interconnected by both ISC and ICB. In this example, each server contains a processor/memory cage and a single I/O cage; G5 servers support two additional I/O cages. The processor/memory cage contains a multichip module (MCM) that has the processors (PU), L2 cache (L2), and I/O hub chips (Hub). The largest G5 processors have four hub chips. Each hub chip has six self-timed interfaces (STI), providing a maximum of 24 STIs per server. The STI was originally designed to be an interface to system memory, and in the G3 and G4 servers, STIs were used only to connect the processor/memory cage to the I/O channel adapters in the I/O cages. In G5, STIs are still used to connect I/O channel adapters, such as ISC, ESCON*, and parallel channels, and open system adapters (OSA) [5], but additions to the hub chip and microcode transform STIs into ICBs, a new S/390 channel interface.

The I/O cages in Figure 1 show the connections to typical S/390 I/O channel adapters, in this case ISCs. The STI from the hub chip in the processor/memory cage connects to a bridge chip in the I/O cage. This chip provides internal buses (IBs) that connect several I/O channel adapters, including the ISC. Each ISC element provides two ISC interfaces, one of which is shown as an interconnection between Server A and Server B.

ISC: An example of a typical S/390 I/O channel adapter

In order to better understand the function of the ICB and how the design approach was chosen, it is important to describe the ISC. This description will reveal the nature of S/390 Parallel Sysplex traffic between instances of OS/390 and the coupling facilities, the function of a typical S/390 I/O channel adapter, and the operation of the STI.

S/390 I/O channel adapters such as the ISC are direct memory adapter (DMA) engines which move data from the peripherals to and from system memory without direct involvement of any of the processors. They also provide memory protection by protecting peripherals from storing into any arbitrary location in memory. The peripherals can only store into those memory locations known to the I/O channel adapters, and those locations are determined by the programs running in the processors. I/O channel adapters provide error isolation to keep errors on the I/O interface from propagating into the system. The I/O channel adapters are considered a trusted, integral part of the server. The traffic between the I/O channel adapter and the peripheral is said to use channel semantics; that is, the data are identified by the operation, and no memory addresses are transferred.

- *ISC link protocols*

Parallel Sysplex passes messages between instances of OS/390 and coupling facilities by using a command/response protocol. Messages from instances of OS/390, called primary messages, are sent by the Send Message instruction, and have optional read or write data; messages from the coupling facility, called secondary messages, have no data. The ISC was specifically designed to transport this information; **Figure 2** shows the basic exchanges over the ISC interface. The ISC on the OS/390 side of the link is called a sender ISC, and the ISC on the coupling facility side is called a receiver ISC. Sender ISCs send primary messages to receiver ISCs, while receiver ISCs send secondary messages to the sender ISCs. Message command frames are called message command blocks (MCBs), and response frames are called message response blocks (MRBs); both contain up to 256 bytes of information. If there are no data to be transferred, the message exchange simply consists of the command, an MCB frame, followed by the response, an MRB frame.

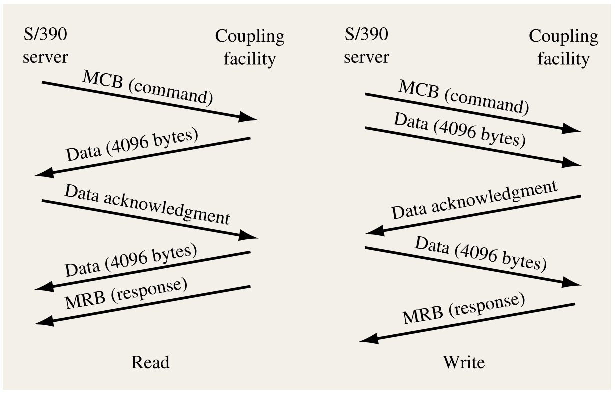

Figure 2

Message exchange protocol for ISC interface.

There is only one round trip in the exchange. Data frames are added to the protocol to exchange more information. Each data frame can have up to 4096 bytes of information, and if only one data frame is required, the exchange still uses only one round trip. When multiple data frames are required, they are paced by data acknowledgment (Data Ack) frames that add extra round trips on the link. Both the read and write examples in Figure 2 show two data frames, and both examples require two round trips on the link. Each message exchange requires hardware resources called buffer sets that contain separate areas for MCBs, MRBs, and data; to improve link utilization, multiple buffer sets are provided to exchange multiple messages over the link at the same time.

The ISC protocol reduces latency at relatively long distances by using a flow control that minimizes the number of round trips, but this protocol requires buffers in the ISC that are large enough to receive an entire MCB and data frame for each of the multiple messages, or buffer sets, on the link. For example, two buffer sets, each with 4096-byte data frames and 256-byte command and response frames, require 8704 bytes of buffering.

The secondary messages sent from the coupling facility to instances of OS/390 manipulate local cache and list (or queue) validity vectors [6]. The ISC microprocessor at a sender ISC receives these secondary messages, manipulates the vectors, and sends the response back to the coupling facility. These messages have no data and consist of MCB/MRB pairs. Only 256 additional bytes of inbound buffering are required to process these messages. A third type of secondary message is used to send fencing commands, and these commands are forwarded by the sender ISC microprocessor to one of the system processors, the system assist processor (SAP), for execution. The ISC microprocessor also performs link

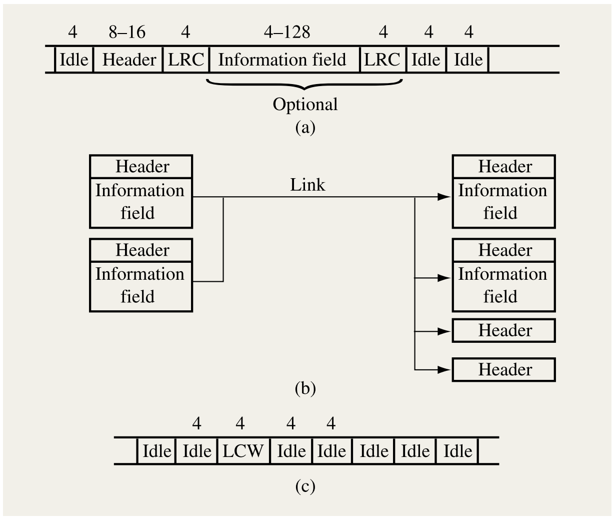

Figure 3

STI logical layer: (a) Packet format; (b) packet buffers; (c) link control word format.

and I/O channel adapter initialization and recovery from hardware and link errors. The error rate of optical transceivers requires retransmission of damaged frames to improve throughput by reducing the number of instances of more global and therefore much slower recovery.

The ISC, like other S/390 I/O channel adapters, communicates with the rest of the system over STI links. Much of the STI traffic consists of memory requests for fetching and storing data. The data are then transmitted and received over the ISC link, as described above.

Self-timed interface

Introduced on the G3 server, the S/390 self-timed interface (STI) uses advances in the density and speed of CMOS chip technology to implement a new system interconnect based on relatively narrow and fast point-to-point links. These links are direct connections between CMOS VLSI chips and do not require any external cable driver circuits. Other examples of similar links include Tandem's system area network (SAN) called ServerNet† [7], the IBM RS/6000* SP2* High-Performance Switch [9], and the IBM RS/6000 SP2 Switch [10]. These links depart from traditional bit-serial designs capable of relatively long-distance communications. They also depart from the very short-distance and wide data buses such as those connecting processors to their L2 caches. Instead, these links use narrow parallel data buses at medium distances, up to tens of meters.

The STI consists of three layers: physical, logical, and user. Both I/O channel adapters and ICB use the two lower layers, the physical and logical, while the highest layer, the user, differs between the two. The physical layer is the attachment to the interface, and it has ten independent differential signal pairs in each direction (dual simplex). Eight of the signals are each a byte of data, and the ninth is a combination parity and tag line. The tenth signal is a half-speed clock; data are sampled on both the rising and falling edges of this clock. The physical layer also performs the self-timing of the interface. The skew between the signals introduced by the differences between the individual cable conductors, the circuits driving and receiving the signals, and the wiring of the chip can exceed the clock period of 3 ns (at 333 MB/s). With this much skew, the clock signal cannot be used directly to coherently sample the data signals. The physical layer time-adjusts individual data and parity/tag signals with respect to the clock, using dynamically adjustable electronic delay lines [11].

The middle STI layer, the logical layer, uses packets to transfer information between the two endpoints of the link. Each packet, shown in **Figure 3(a)**, has a header (8 or 16 bytes) and a header checking field (longitudinal redundancy check, LRC). Packets may also have an optional information field, up to 128 bytes, and a second LRC to protect the information field. Packets are temporarily stored in the buffers [**Figure 3(b)**]. In each direction there are two transmit packet buffers (shown on the left) and two receive packet buffers (shown on the right). Two additional receive header buffers provide a path for certain control information to avoid deadlocks that may otherwise be caused by the higher-level layers. An identical set of buffers (not shown) is provided for traffic in the opposite direction. STI packet flow control is "credit-based," which means that packets are sent only when there is sufficient buffer space at the receiver, and these credits are communicated back to the transmitter by link control words (LCWs). LCWs [**Figure 3(c)**] are four bytes long and may be inserted anywhere in the data stream, including within packets. Each time a packet header is successfully received, the STI logical layer sends an acknowledgment LCW back to the transmitter. Using time-out timers and a one-bit sequence counter, packets with damaged headers are detected and automatically re-sent by the logical layer. A second acknowledgment is sent back to the transmitter when the user layer reads the packet from the receive packet buffer. It is then the responsibility of the user layer to resend packets with damaged information fields. The maximum data rate over the link can be achieved only if packets are sent back to back. With the STI credit-based flow control, two packet buffers allow two exchanges on the link at the same time, achieving the maximum data rate at link distances of a few tens of meters.

The original third STI layer, called the native STI user layer, has memory semantics and is used by all I/O channel adapters such as ISC to directly access system memory. The basic operation of an I/O channel adapter is to send memory requests. Each memory request receives a response. For a memory fetch, the memory address is in the request, and the response contains the data. For a memory store, the memory address and the data to be stored are in the request, and the response simply contains an indication of the success of the store operation.

Figure 4(a) shows the STI header for I/O channel adapter memory requests. The first eight bytes contain mostly STI logical layer controls and routing information, and the second eight bytes contain the memory command. Included within the command are the memory op code (store or fetch) and the memory address. **Figure 4(b)** shows the STI header for I/O channel adapter memory responses, where the second eight bytes of the header contain the memory response codes, one of which is used to indicate the successful completion of a memory operation, while other codes indicate failures such as an invalid memory address.

During an I/O input (read) operation, the I/O channel adapter receives data from an I/O interface (such as the ISC) and typically buffers this information. Memory addresses and control information within the I/O channel adapter are used to generate the STI headers, and the buffered data is placed in the information field. Each request packet sent to memory requires a response packet containing the response code, as shown in **Figure 4(c)**. During an I/O output (write) operation, the I/O channel adapter is instructed to send data over the I/O interface. As in input operations, memory addresses and control information within the I/O channel adapter are used to generate the STI headers. Header-only packets containing the op code and memory address are sent from the I/O channel adapter, and the response contains both a header and the requested data in the information field. Again, each request packet sent to memory requires a response packet containing not only the data but also the response code, as shown in **Figure 4(d)**.

Additional packet exchanges allow I/O channel adapters to manipulate interrupt and busy bits in the hub chip. Processors can also initiate packet exchanges, either to send eight bytes of control information to the I/O channel adapters or to sense eight bytes of information from the I/O channel adapters. The ISC uses the interrupt and busy bits to signal the completion of work, and the processors use the control commands to prepare the ISC addresses, initiate messages, and log error information. We describe below how the ICB appears to the processors to use the same commands as those used by the ISC.

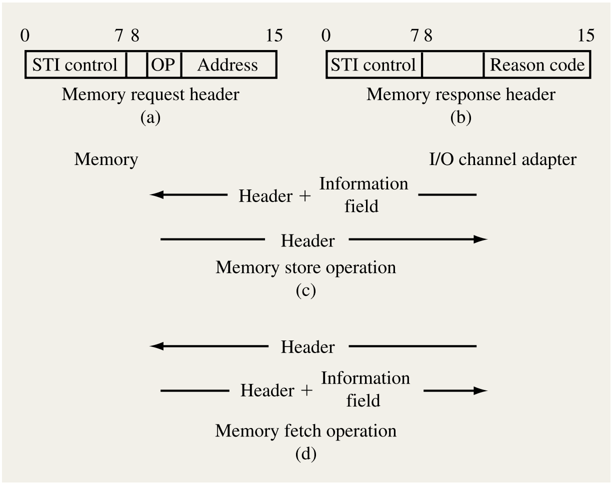

Figure 4

Native STI user layer: (a) Memory request header; (b) memory response header; (c) I/O input (read) operation requiring a memory store; (d) I/O output (write) operation requiring a memory fetch.

ICB

The most basic concept for the ICB was to take advantage of the high bandwidth and low latency of the STI to pass messages more directly in a Parallel Sysplex while keeping the upper interfaces common with the ISC. Certainly, OS/390 and coupling facility instances see no differences between ISC and ICB other than that they are different channel types with different performance capabilities. The same instructions are used for both ISC and ICB, and the microcode that implements these instructions has only very small differences between the two. However, the microcode to initialize and recover the ICB is much different from that for the ISC.

As described in the previous sections, the native STI user layer requires a function similar to that of an I/O channel adapter to drive memory requests and actually move the data. One of the many implementations we considered was to develop a new chip that would look like a channel-to-channel I/O adapter, that is, a pair of ISCs attached back to back. This chip would connect two or more hub chips and would drive the memory requests to all attached servers. The major advantage to this approach was that native STI user-layer protocols could be used, and no change to the hub chip would be required. On the other hand, having a single component that is an integral part of multiple servers posed some severe problems. Where would such a chip be packaged, which server would be responsible for its maintenance, and how could errors be isolated to a single server? Also, adding a new chip adds to the product cost, slows the performance, and

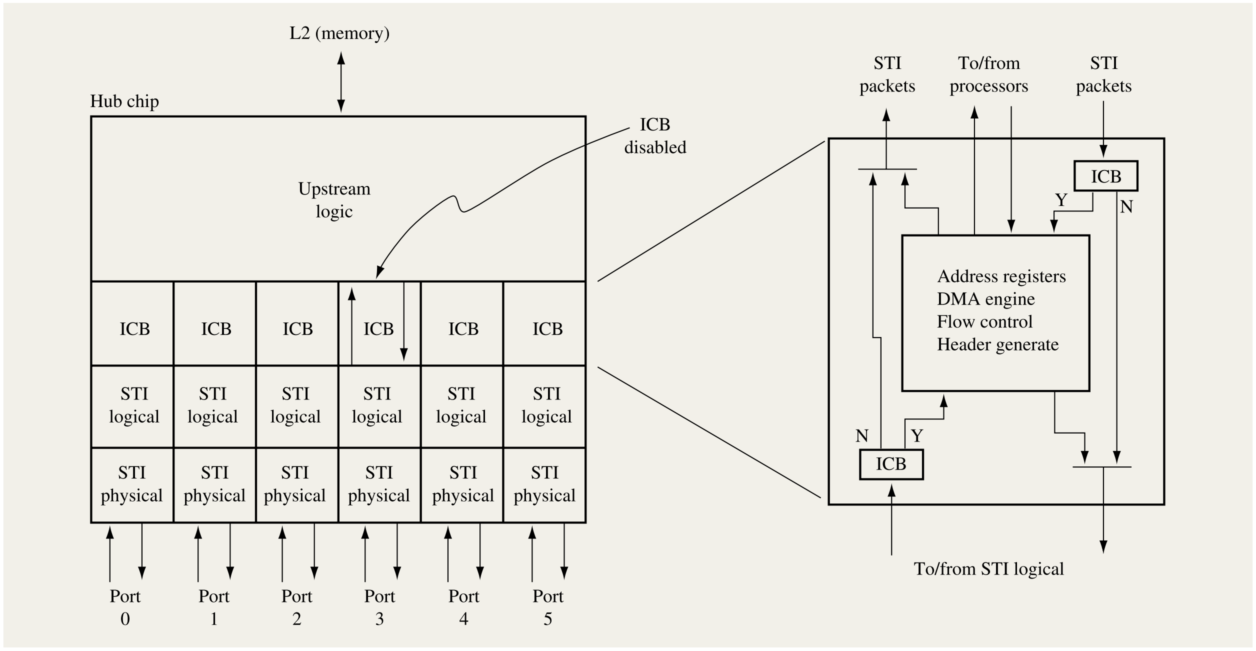

Figure 5

Hub chip and ICB.

decreases reliability, and that is in direct conflict with the design goals of the ICB. We decided that the only workable solution was to put the channel function into the hub chip.

Since adding new chips was not an option, we set out to determine whether we could design an "ISC" channel adapter compact enough that six would fit in a hub chip while requiring only very small changes to the rest of the chip. The design not only had to be compatible with the upstream hub chip logic, but also had to use the logical and physical STI layers without any changes. Within the restrictions placed on the design by the STI logical layers, we had to develop a whole new channel interface architecture using a newly defined set of STI packets. All of the features and design considerations of any other S/390 I/O channel adapter had to be addressed, including initialization, recovery, and error isolation at both the ICB and the link levels. There was obviously no space for any microprocessors, so the entire mainline protocol had to be handled by hardware state machines, and we had to find an alternative way to process secondary messages. We added six copies of the new ICB logic, one for each STI port, between the STI logical layer and the upstream hub logic, as shown on the left side of **Figure 5**. When the STI port operates in native mode with I/O channel adapters attached, the ICB function is disabled, and all packets flow directly between the upstream hub logic and the STI logical layer. When the ICB is enabled, the new set of packets defined by the ICB user layer is transferred over STI. These new packets are intercepted by the ICB channel, as shown on the right side of Figure 5. To allow the processors to communicate with the ICB, a path was added to send it the control and sense commands, similar to those which are sent over STI to ISC.

- *ICB packets and DMA*

Packets defined by the ICB user layer have channel semantics rather than memory semantics. The headers contain only eight bytes each, and most of the bits are controls required by the STI logical layer. Some of the unused bits in the headers are used to indicate the type of message area (MCB, MRB, or data), the buffer set number, a sequence count, a continue bit, and a new flow-control packet type which is described later. Also, the flow of packets does not consist of the request/response pairs used by I/O channel adapters in the native STI user layer. Instead, multiple command, data, and response packets are streamed together to transmit MCBs, MRBs, and data areas. The data rate achieved by ICB packets is higher than that of native STI user-layer packets because ICB headers are shorter by eight bytes and because a response is not required for each individual ICB packet. The ICB request/response protocol is defined on a message basis.

The flow of packets is controlled by the ICB DMA and is shown in **Figure 6**. After a processor sends a command to start a message, the ICB generates a memory fetch command that has the same format as native STI user-layer commands sent from the STI logical layer to the upstream hub chip logic. The memory addresses and count information used in the memory fetch commands are generated from addresses and control registers within the ICB (some of them are contained in the command from the processor). The rest of the information required to generate the memory commands is set by mechanisms described later. The memory fetch command is sent to the upstream hub logic, and when the response is received, the reason code is checked to determine whether the memory fetch was successful. If it was, the 16-byte header of the memory response packet is discarded and a new eight-byte ICB header is generated. The new ICB packet sent over the link consists of the new header and the information field from the memory response packet.

When the ICB receives a packet from the STI logical layer, it examines the header to see whether it is an ICB packet or a native STI user-layer packet. Native user-layer packets are used to signal more primitive conditions from one server to another, and they are described below. If it is an ICB packet, the ICB examines the eight-byte header to determine the type of packet (MCB, MRB, data area) and the buffer set number. The eight-byte header is discarded, and a 16-byte header with the native STI user-layer format is generated by ICB. From information in the original eight-byte header, a memory store packet is generated. The memory addresses and count information used in the memory store commands are generated from address and control registers within the ICB, and the new packet consists of the new packet header and the information field from the received ICB packet. When the response is received, the reason code is checked to determine whether the memory store was successful. If it was, the response packet is discarded and not sent over the link.

Since multiple packets are required for all but the shortest MCBs and MRBs, the ICB generates multiple sequential memory fetch requests. To conserve bits in the STI header, counts are not used. Instead, a single bit in the header identifies the end of the message area. Since the STI logical layers guarantee delivery of all packet headers, a single bit is all that is required.

- *Elimination of ICB buffers*

One of the largest users of chip real estate in the ISC is the group of high-speed receive-frame buffers. It was impossible to fit almost nine thousand bytes of high-speed buffering for each of the six ICBs on the hub chip. We knew that the flow control and packet buffers of the STI logical layer could help us eliminate these buffers, but we

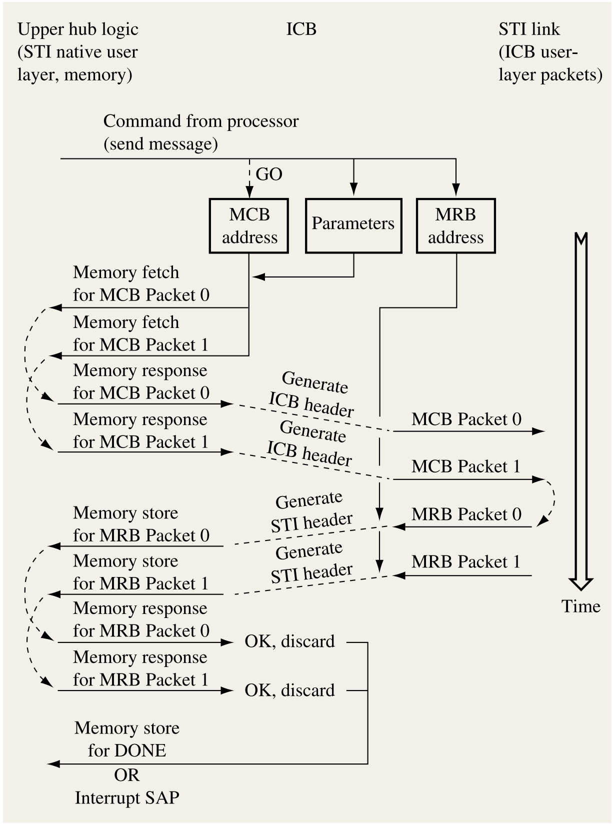
Upper hub logic (STI native user layer, memory)
ICB
STI link (ICB user-layer packets)
Command from processor (send message)
GO
MCB address
Parameters
MRB address
Memory fetch for MCB Packet 0
Memory fetch for MCB Packet 1
Memory response for MCB Packet 0
Generate ICB header
MCB Packet 0
Memory response for MCB Packet 1
Generate ICB header
MCB Packet 1
Memory store for MRB Packet 0
Generate STI header
MRB Packet 0
Memory store for MRB Packet 1
Generate STI header
MRB Packet 1
Memory response for MRB Packet 0
OK, discard
Memory response for MRB Packet 1
OK, discard
Memory store for DONE OR Interrupt SAP
Time

Figure 6

ICB packet flow.

still needed a way to keep received packets moving through the ICB logic without waiting for a processor to calculate memory addresses. For example, if the ICB did not have the memory addresses needed to generate memory store requests for all packets as they were received, and instead it had to interrupt a processor to calculate the address, the entire link would stop, waiting for a processor to respond to the interrupt before the packet information field could be sent to memory as described above. In early implementations of the ISC, waiting for a processor yielded reasonable performance, since frames are buffered and the traffic on the link does not stop. However, in the ICB, after the first two packets are received (256 bytes), the receive buffers in the STI logical layer are full, and all traffic on the link must stop while waiting for these packets to be moved to memory. We realized that in order to keep the STI link running at top speed, every received packet must be processed immediately by the ICB. This means that the memory addresses for all received packets must be known to the ICB prior to their arrival.

We next examined every type of ICB packet received on the link. Primary MCBs received by coupling facilities (receiver ICBs) could be sent to memory immediately, since the addresses for these packets are prepared by the coupling facility code and are known to the ICB. Likewise, when the coupling facility receives responses to its secondary messages, the memory addresses are already established. This design point was followed in all ISC implementations except for the very earliest prototype hardware. On the OS/390 side (sender ICBs), we found problems with receiving both MRBs for primary messages and MCBs for secondary messages. In both cases, the ISC design depended on the receive buffers for temporary storage of the frame, while processors, sometimes the ISC microprocessor, were interrupted to handle them. Finally, we found that handling data area frames required an entirely new design.

Two different design approaches were developed to ensure that the ICB had the memory addresses before the packets were received. The first is to send the memory address from a processor to the ICB before the packet is received; the second is to delay transmission of packets until the memory addresses have been calculated. The first approach to preparing for packet arrival, originally designed for the ICB, was also applied to the ISC [5]. As will be seen, use of this approach for both ISC and ICB not only improves the ISC performance but is also the greatest factor in the commonality between the microcode and the coupling facility control code. The second approach is unique to the ICB, and it adds a link exchange to start data transfer for messages requiring read or write data.

In the early designs, a processor sent commands to the ISC instructing it to send a primary MCB. Later, when the MRB was received, it was put in a receive buffer, and a processor was interrupted to generate a command to the ISC to move the MRB to memory. In the ICB, without buffers, this meant that the link would stop while the processor detected the interrupt and moved the MRB to memory. The solution was to develop a new command from the processor that combined the functions of the command to send the MCB and the command to receive the MRB. The new command causes the ICB to send the MCB, and the command includes the memory address of the MRB. The ICB is therefore always prepared for the reception of the MRB before it arrives. This new command was also retrofitted into the ISC to reduce latency, improve performance, and allow more common microcode between ISC and ICB.

Early ISC designs also required processor involvement in moving data areas for primary messages. During read operations, when a data frame was received by a sender ISC, a processor was interrupted to move the data frame from the ISC receive buffer to memory. Additional signaling to the ISC microprocessor was required when the data move to memory was successfully completed. Write operations at the sender ISC required a processor to send individual commands to move each data area from memory over the link. Interrupts to a processor were used to signal that the data area was successfully received by the receiver ISC. Similarly, the coupling facility code had to move each data area individually. Moving 16 data areas required the use of a processor at both ends of the link 16 times.

To solve this problem, hardware was developed to move multiple data areas whose memory addresses are stored in a list in memory. At the sender ICB, if the processor is to move one or more data areas, it generates a list of memory addresses and sends the ICB a new command containing the memory address of the list, along with parameters including the direction of data transfer, the number of data areas to transfer, and the lengths of the data areas. When the ICB receives the list processor command, it fetches the first address in the list and moves the data area. After the count has been exhausted, the next address of the list is fetched, and the data area for this address is moved. At the receiver ICB, the same list processor is used, but is invoked differently. After the coupling facility control code decodes a received message and determines that data areas are to be moved, it generates the list of addresses and issues a special instruction to move the data. The microcode that implements this instruction generates the command, including the address of the list, and sends it to the ICB. The ICB fetches addresses and moves data areas in the same way as described previously. The list processor was such a performance improvement for multiple data area moves that it, like the new command to prepare for the receipt of the MRB, was also added to the ISC design. Changes in both the microcode and coupling facility code from the previous way of handling multiple data areas were quite extensive, so adding the list processor to the ISC was significant in creating code commonality between the ISC and the ICB.

However, the list processor alone did not solve all of the design problems for the ICB. Consider a message containing write data areas. In the ISC, the first write data area is sent immediately after the MCB and is temporarily stored in a receive buffer. Only after the coupling facility code processes the MCB can the list of addresses be generated to move any of the data areas to memory. Typically, it takes several microseconds for the coupling facility code to generate this list. In the ICB, if the first few data area packets were to be sent immediately after the MCB, the link would stop, since the memory addresses for the data area would not yet have been received by the ICB. A similar but much less severe problem exists for messages with read data. In this case, the processor executing the Send Message instruction may experience enough delay between sending the command to move the MCB and the command for the list processor that the first few data area packets are received and stop the link until the list processor command is sent to the ICB.

To solve the data area start-up problem, the ICB added a new packet, consisting of only an eight-byte header, called a data request packet. This packet creates a new exchange on the link, but at ten meters, the added latency, and therefore the loss in performance, is negligible. During messages containing write data areas, the sender ICB is delayed until a data request packet is received. At the receiver ICB, when the coupling facility code issues the instruction to move the data areas, the ICB sends the data request packet when it receives the list processor command. Data packets are thus transmitted only when the receiving end of the link is prepared with the memory addresses. Similarly, when messages with read data areas are sent, the receiver ICB waits until it receives a data request from the sender ICB before sending data area packets. When the sender ICB receives the list processor command, it sends a data request to the receiver ICB. Only when both a data request packet and a list processor command have been received does the receiver ICB start transmitting packets. Once data transfer starts, it continues from list entry to list entry without any additional protocols on the link. Recall that the ISC requires link data acknowledgments between multiple data areas. Without link acknowledgments, the sizes of the data areas specified by the list parameters may be different, and scatter/gather operations are transparent to the link. An example of a scatter operation is a write command from an OS/390 instance to a coupling facility in which a contiguous data area in the OS/390 instance memory is scattered into several smaller areas in the coupling facility memory. A gather operation is just the opposite: Several data areas in the coupling facility memory are gathered into a single contiguous area in the OS/390 instance memory during a read operation.

All of the features described above allow the ICB to be implemented without receive buffers. The ICB takes advantage of the limited distance of the link to tolerate both the low-level STI logical layer link acknowledgments and the added exchange of the data request packet.

- *Secondary messages*

Fast processing of secondary commands, particularly cache invalidation commands, is important to Parallel Sysplex performance, and one of the functions of the ISC microprocessor is to execute these commands. Since ISC microprocessor performance has not improved with new CMOS server generations, this microprocessor becomes a less attractive way of processing secondary commands as the server processors become faster.

In the ICB, we reengineered secondary command processing. Since the ICB is part of the hub chip, secondary command performance automatically improves, or scales, with new server generations. Adding a microprocessor to the ICB was clearly not an option because of the chip real estate required. Another option we investigated was adding a hardware state machine to the ICB for processing these commands. It quickly became apparent that such a state machine was far too complex to attempt. It also required a fair amount of chip real estate to temporarily store the MCB, MRB, and address and state information, but it was smaller than adding a microprocessor. At the same time, we still needed a mechanism to transmit the fence commands to a SAP.

Instead of adding a microprocessor or a special state machine, we considered using the processors (PUs); obviously, their performance scales. First, we had to determine whether the level of processor cache disruption was acceptable. Since secondary message processing is not very complicated, the cache effects were negligible. Next, the ICB needed memory addresses to move the secondary MCBs to memory and to fetch the responses. Since secondary commands are received only by sender ICBs, we used the same hardware facilities used by the receiver ICBs to receive primary messages and send their responses. Finally, we needed a rapid way to interrupt a processor. One of the proposals was to broadcast the interrupt to multiple processors. This way, the least busy processor could handle the interrupt. Unfortunately, the code path length and the processor cycles consumed in overinitiative made this proposal unworkable. We decided on a design that assigned interrupts from each individual ICB to a processor. Only one processor handled the hardware interrupts for a particular ICB, and the ICB workload could be distributed evenly among all available processors.

At the end of each S/390 instruction (end-op), the processors determine whether there are any hardware interrupts from the ICBs. Since the synchronous version of the Send Message instruction may take a relatively long time to execute (longer than any other instruction), we made it interruptible from the time the commands are sent to the ICB until the time at which the message response is received from the coupling facility. If hardware interrupts are pending, the processor looks at the MCB, already moved to memory by the ICB, and determines whether it is a cache invalidate or list notification message. If it is, the processor executes the command, builds the response in memory, and sends a command to the ICB signaling it to send the MRB. If the secondary message is a fencing command, initiative is passed to a SAP.

- *SAP-to-SAP packets*

The ICB packets described above (MCB, MRB, data, and data request) are not in themselves sufficient to implement the ICB. A separate group of packets called SAP-to-SAP packets allow additional signaling and are required for initialization and recovery protocols. These packets allow a SAP in one server to set a group of SAP interrupt bits in the hub chip in the other server. As shown on the right side of Figure 5, when the ICB receives these packets, it determines that they are not ICB packets and passes them directly to the upstream hub chip logic. Similarly, when these packets are sent down from a SAP, they are sent directly to the STI logical layer for transmission on the link. Specific uses for these signals are described below.

- *STI link initialization in ICB mode*

The STI physical layer was originally designed to operate in a master/slave mode with the hub chip as master and the bridge chip as slave. The ICB operates in a peer mode, in which both ends of the link are hub chips, and this mode caused us to substantially redesign the link initialization process. For example, since each end of the ICB is in a different server, and the power for each is independent, the initialization microcode had to be changed to be tolerant of a powered-down server. When the other end of the link powers up, the STI physical layer detects this condition, and the ICB initialization process is restarted.

When STI links are connected to bridge chips, the hub chip sends sense commands to determine the topology of the I/O subsystem; the hub chip sends the sense command and receives the response. In the ICB, this sense command tells the initialization code that hub chips are interconnected and are therefore candidates for becoming an ICB link. Hub chips were always capable of sending the sense commands, but changes had to be made to allow them to respond to these sense commands. The initialization microcode also had to be changed to recognize this new response code from a hub chip.

- *ICB initialization*

The absence of message buffers in the ICB also affects the way that packets are handled during ICB initialization. When an ICB is initialized, several SAP-to-SAP packets are exchanged to synchronize the initialization process at both ends of the link. For example, both ends must agree that the ICB is active and ready to receive initialization packets. In preparation for the exchange of initialization packets, a SAP loads their hardware system area addresses into the ICB. The initialization packets exchange node descriptor information, and the format of these packets is the same as that for MCBs and MRBs. The initialization packets include indicators that the node is either a sender or a receiver ICB. When an initialization packet is received, the ICB moves it to the address in the hardware system area, and raises an interrupt to a SAP. The SAP examines the node descriptor information, replies by building a node descriptor response in a hardware system area, and commands the ICB to send it over the link.

- *Unprepared buffer sets*

Sometimes one or more primary-message buffer sets in the receiver ICB are not prepared to receive messages. This condition exists when the coupling facility code is not yet initialized, or when a buffer set is still recovering from a previously detected error. In either case, the MCB target address in coupling facility memory either is not known because the coupling facility code has not yet started, or is unavailable because the coupling facility code has not yet processed the previous error. In these cases, an alternate memory address is used. When an MCB is received for a buffer set that is not prepared, the ICB moves it to the alternate address in a hardware system area and interrupts a SAP. This is the same memory location as that used in initialization mode. If the SAP determines that the coupling facility cannot process the message, it sends a special MRB response back to the sender ICB. If the SAP determines that the coupling facility code has processed the previous error and is able to process the message, the SAP moves the MCB from the hardware system area to the coupling facilities' buffer set area in memory. It then prepares the ICB memory address for subsequent messages and alerts the coupling facility.

RAS features

Like any other S/390 channel adapter, the ICB requires comprehensive error detection and recovery, essential in achieving a high level of reliability, availability, and serviceability. All of the restrictions that come with our design approach, that is, making as few changes to the hub chip as possible, make error handling particularly challenging. For example, we had to use the error-detection and recovery mechanisms provided by the STI logical layer and the upstream hub chip logic while adding checking of our own that is unique to the ICB.

The STI logical layer detects all errors associated with packet delivery, and in some cases it automatically resends packets with damaged headers. While this function performs useful error recovery, it makes the ICB design more complicated, since packets are not necessarily received by the ICB from the STI logical layer in the same order that they are transmitted. To detect this condition, we added a sequence count in the ICB packet headers. When the ICB receives a packet with the wrong sequence number, it assumes that the correct packet is being retried and simply waits for the next packet to be received. The next packet received must have the correct sequence number.

Another level of recovery is triggered by a message time-out. All messages are timed by hardware timers in the ICB hardware, and when a time-out condition is detected, an interrupt is raised to a SAP. Because of the out-of-order packet delivery caused by STI link errors, the ICB must perform link synchronization before presenting the interrupt to a SAP. This process suspends the sending of any new packets until all outstanding packets have been acknowledged. When all of the acknowledgments have been received, all STI logical layer packet retries have been completed. When the ICB sends the time-out interrupt to the SAP, the microcode knows that no "stale" packets are on the link. A buffer-invalidate sequence is then performed using SAP-to-SAP signaling. Again, the STI guarantees delivery of all packets, making the invalidate process simpler than in the ISC.

The STI logical layer also detects errors that cannot be recovered by resending packets. These include errors in the packet information field, loss of synchronism between the clock and data bits, and protocol errors associated with packet flow control, such as receiving multiple acknowledgments for the same packet. The errors that affect all buffer sets are reported directly to a SAP, while errors that can be isolated to a single buffer set are intercepted by the ICB.

The ICB performs additional protocol checking unique to its user-layer packet definition. One kind of error is the receipt of packets that are not defined by the ICB, such as data area packets whose information fields do not contain exactly 128 bytes. A more complex form of error is the receipt of a packet at the wrong time. Each buffer set has state information that describes the progress of its message exchange, and each received packet is checked with respect to this state. Also, the ICB updates these states as frames are both sent and received. For example, if a buffer set has started a message for a read operation and a data request packet is subsequently received, a data request packet protocol error is detected. In read operations, the data request packet should have been sent from the reading side of the link, not the receiving side. In general, most of these protocol errors indicate a design error, and the recovery microcode logs as much information as possible to assist in problem determination.

The ICB also detects a few sequencing errors from the processors. These errors are always caused by design errors and are very difficult to isolate without these simple error checkers. One example is the so-called "double" error. If a processor sends the same command twice before the message exchange completes, a double error is detected.

Finally, the ICB intercepts errors detected by the memory subsystem and reports them to a SAP. This is in contrast to native STI attachments, where memory subsystem error responses are passed along to the I/O channel adapter. Memory errors include uncorrectable errors found in the memory when fetching data, memory addresses that do not exist, and invalid memory op codes. With the exception of the uncorrectable memory error, these errors usually point to a design problem.

- *Trace*

Dedicated trace hardware provides important information for debugging problems. Having dedicated hardware allows the trace function to operate all the time and does not adversely affect performance. Microcode trace schemes are also extensively used, but they slow performance, so developers usually remove as many trace points as possible as the design becomes more stable. The ICB trace hardware is a multiple circular buffer, tracking the progress of each of the four buffer sets independently. Each buffer set has eight entries, and events such as packets sent and received, commands from the processors, and memory requests and responses can all be traced. Each entry includes a time stamp. Many design errors affect only one buffer set, causing it to stop in the middle of a message exchange while the other buffer sets continue to operate. For this most common type of design error, a single trace area is not as useful as having independent areas, because the events for the buffer sets without the error consume all of the trace space, overwriting the most useful information (that pertaining to the stopped buffer set).

Conclusions

Parallel processing beginning with symmetrical multiprocessors (SMPs) has always exploited higher bandwidths and lower latencies. The better the memory subsystem, the better the SMP performance. Parallel processing with clusters of SMPs is also starting to exploit the high bandwidth and low latencies provided by self-timed bus technologies, and Parallel Sysplex, the S/390 cluster offering, has the ICB for its new interconnect. The ICB offers great cost and reliability benefits by having an optimized hardware structure that does not require any additional chips in the processor MCM, and performance studies show that the resources required from the processors are small. The ICB will naturally scale with future generations of processors as chip and STI link speeds increase.

Acknowledgments

The ICB evolved over several years and required the efforts of many workers in the Poughkeepsie, Boeblingen, and Endicott laboratories. Special acknowledgment goes to Lothar Klein and Elke Nass for their microcode skills, and to Norbert Schumacher for his hub chip design and simulation work. We also thank Audrey Helffrich, Patrick Sugrue, and Walter Von Dehsen for their review of this paper.

*Trademark or registered trademark of International Business Machines Corporation.

†ServerNet is either a registered trademark or a trademark of Tandem Computers Incorporated in the U.S. and/or other countries.

References

1. J. M. Nick, B. B. Moore, J.-Y. Chung, and N. S. Bowen, "S/390 Cluster Technology: Parallel Sysplex," *IBM Syst. J.* **36,** No. 2, 172–201 (1997).
2. J. M. Nick, J. Chung, and N. S. Bowen, "Overview of IBM System/390 Parallel Sysplex—A Commercial Parallel Processing System," *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, 1996, pp. 488–495.
3. C. L. Rao, G. M. King, and B. A. Weiler, "Integrated Cluster Bus Performance for the IBM S/390 Parallel Sysplex," *IBM J. Res. Develop.* **43,** No. 5/6, 855–862 (1999, this issue).
4. C. L. Rao and C. Taaffe-Hedglin, "Parallel Sysplex Performance," *CMG Trans*. No. 87, pp. 3–7 (Winter 1995).
5. T. A. Gregg, "S/390 CMOS Server I/O: The Continuing Evolution," *IBM J. Res. Develop.* **41,** No. 4/5, 449–462 (July/September 1997).
6. IBM Corporation, *MVS/ESA Programming: Sysplex Services Guide*, Order No. GC28-1495-02, June 1995; available through IBM branch offices. Chapter 6 describes coupling-facility cache structures, Chapter 7 describes list structures, and Chapter 8 describes lock structures.
7. R. W. Horst, "TNet: A Reliable System Area Network," *IEEE Micro* **15,** No. 1, 37–45 (February 1995).
8. W. E. Baker, R. W. Horst, D. P. Sonnier, and W. J. Watson, "Flexible ServerNet-Based Fault-Tolerant Architecture," *Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing. Digest of Papers—International Symposium on Fault-Tolerant Computing*, 1995, pp. 2–11.
9. IBM Corporation, *The SP2 High-Performance Switch—SJ34-2*, Order No. G321-5564; available through IBM branch offices.
10. *IBM POWERparallel Technology Briefing*, URL: *http://www.rs6000.ibm.com/resource/technology/sp_sw2/spswp2_1.html*.
11. J. M. Hoke, P. W. Bond, T. Lo, F. S. Pidala, and G. Steinbrueck, "Self-Timed Interface for S/390 I/O Subsystem Interconnection," *IBM J. Res. Develop.* **43,** No. 5/6, 829–846 (1999, this issue).

Received January 4, 1999; accepted for publication June 3, 1999

Thomas A. Gregg *IBM System/390 Division, P.O. Box 950, Poughkeepsie, New York 12602 (tomgregg@us.ibm.com).* Mr. Gregg is a Senior Technical Staff Member in the S/390 System Design group. He received an SC.B. degree in engineering from Brown University in 1972 and continued under a university fellowship, receiving an SC.M. degree in electrical engineering in 1974. He joined IBM at the Poughkeepsie Laboratory in 1973. Mr. Gregg has held various technical positions in the area of I/O subsystem design. He holds numerous patents utilized in IBM ESCON and Parallel Sysplex channel products, and has received nine IBM Invention Achievement Awards. He received an IBM Corporate Award and an IBM Outstanding Innovation Award for work on ESCON products, and three IBM Outstanding Innovation Awards for work on Parallel Sysplex. He is a member of the IEEE.

Kulwant M. Pandey *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (pandey@us.ibm.com).* Ms. Pandey is a Senior Engineer working on coupling hardware design. She received a B.S. degree from Birla Institute of Technology, India, in 1973, and an M.S. degree in electrical engineering from Columbia University. Since joining IBM in 1973, she has been involved in hardware design, management, and simulation of cache controllers, channels, and coupling. She has received an IBM Outstanding Innovation Award and several Outstanding Technical Achievement Awards. At present, Ms. Pandey is involved in hardware design of ISC and ICB.

Richard K. Errickson *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (rerricks@us.ibm.com).* Mr. Errickson received a Bachelor of Science degree in computer engineering from Lehigh University and joined IBM in 1985. He is a developer of microcode for the I/O subsystem for S/390 processors.