SPIDER: A HIGH-SPEED NETWORK INTERCONNECT

Mike Galles

Silicon Graphics Computer Systems

The Spider chip, an interconnect for high-end networking applications, sustains a data transfer rate of 4.8 Gbytes/s, either between chips in a single chassis or between remote chassis over cables up to 5 meters long.

SGI's Spider chips—Scalable, Pipelined Interconnect for Distributed Endpoint Routing—create a scalable, short-range network delivering hundreds of gigabytes per second in bandwidth to large configurations. Individual Spider chips sustain a 4.8-Gbyte/s switching rate, connecting to each other and to endpoints across cables up to 5 meters in length. By delivering very high bandwidth—thousands of times higher than standard Ethernet—at low latencies, Spider is ideal for CPU interconnect applications, high-end network switches, or high-performance graphics interconnects.

The Spider chip design drew on the principles of computer communications architecture. Isolation between the physical, data link, and message layers led to a well-structured design that is transportable and more easily verified than a nonlayered solution. Because the chip implements all layers in hardware, latency is very low. Thus, we could realize the benefits of layering without sacrificing performance.

The physical transmission layer for each port is based on a pair of source synchronous drivers and receivers. These SSDs and SSRs transmit and receive 20 data bits and a data-framing signal at 400 Mbps. The data link level, which we call the link level protocol or LLP, guarantees reliable transmission using a CCITT-CRC (standard cyclic redundancy-checking algorithm) code with a go-back-*n* sliding-window protocol[1] retry mechanism. The message layer defines four virtual channels and a credit-based flow control scheme to support arbitrary message lengths. It also defines a header format to specify message destination, priority, and congestion control options.

The chip supports two distinct message types, each with its own routing mechanism. Network administrative messages use source vector routing, while mainstream messages use distributed routing tables that exist in each chip. The routing tables define a static route between every pair of destinations in the network, programmed by software whenever the network is reconfigured. This allows hardware to take advantage of the static route and optimize routing latency. On the other hand, software can step in at any time to define new routes that avoid detected faults or to change existing routes based on new algorithms.

To avoid the blocked head-of-queue bottleneck, a port's receive buffers maintain a separate linked list of messages for each of the five possible output ports for each virtual channel. Each arbitration cycle, the arbiter chooses up to six winners from as many as 120 arbitration candidates to maximize crossbar utilization. To avoid starvation and promote network fairness, messages accumulate a network age as they are routed, increasing their priority.

Physical transmission layer

An SSD/SSR pair provides physical data transmission for each of the six full-duplex ports (see Figure 1 for the chip's block diagram). A single link consists of 20 data bits, a frame bit, and a differential clock per direction. The 200-MHz differential clock is sampled on both edges, making the effective rate of data transmission 400 Mbps. The raw data bandwidth of each link is 1 Gbyte/s per direction.

The chip core operates at 100 MHz, requiring a 4-to-1 serialization between the core and the 400-MHz SSD data pins. At each 100-MHz clock edge, the core provides 80 bits of data, which the SSD serializes into a 20-bit stream over four 400-MHz clocks. The SSD and core use the same clock source with different divisors for synchronous transmission.

The SSR uses the received differential clock to sample incoming data bits. We

0272-1732/97/$10.00 © 1997 IEEE

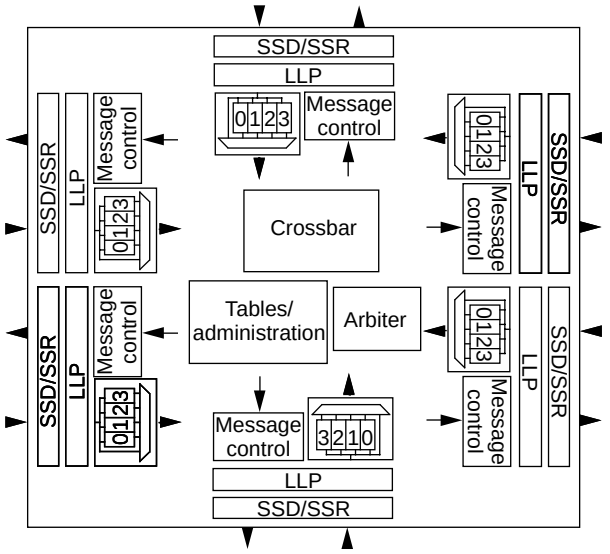

Figure 1. Spider chip block diagram. Each port achieves 2.0-Gbytes/s raw bandwidth link and 1.6 Gbytes/s sustained data transfer. SSD: source synchronous driver; SSR: source synchronous receiver; LLP: link layer protocol.

delayed the differential clock on the board by 1.25 ns to ensure that the clock edge is centered on the desired data sample point across process variation. Once data is clocked into the SSR, it is deserialized and placed into one of two 80-bit registers that serve as the core's receive buffer. The data-framing bit is synchronized to the local core clock frequency, and data is read from the receive buffer into the core.

To prevent receive buffer overruns, we inserted a dead cycle to break up continuous data bursts that exceed a programmable maximum length. This maximum burst length is a function of crystal frequency, and modern crystals have close enough tolerances that burst lengths can be several thousand clocks cycles long.

Data link layer

The data link layer guarantees reliable data transfer between chips and provides a clean interface to the chip core, hiding the physical transmission layer's control details. The interface to the LLP receives and sends data in micropacket quantities (Figure 2), which consist of 128 bits of data plus 8 bits of sideband. Sideband provides an out-of-band channel on the data link to send flow control information, so that sustainable data rates will not be affected by buffer administration. The LLP does not use the 8 bits of sideband for its own protocol layer, but passes them up for message layer flow control.

We chose this relatively small micropacket size because a large class of network transactions are short, fitting into one or two micropackets. Also, we know that data received from a Spider chip is error-free in 16-byte quantities. Thus, endpoints can use partial data from larger transfers immediately, without waiting for a full-message CRC. This is especially important for interprocessor networks, which can implement early pipeline

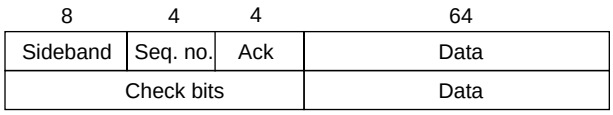

Figure 2. LLP micropacket format.

restart on a partial cache fill. Data protection on a single-link basis instead of end-to-end tolerates message header modification as it flows through the network, allowing such features as message aging and virtual-channel adaption.

Once the LLP receives the data and sideband from the chip core, it assigns a sequence number to the micropacket and stores it in a retransmit buffer. The LLP also appends an acknowledge sequence number to the micropacket and calculates a 16-bit check code using CCITT-CRC. It then transmits the entire micropacket over the link.

If no errors occur, the remote peer acknowledges the sequence number of the transmitted micropacket(s), which the LLP then removes from the retransmit buffer. If an error does occur, the chip uses a go-back-*n* sliding-window protocol for recovery. A go-back-*n* sliding window protocol manages a circular buffer, storing to the head of the buffer during data transmission and freeing from the tail of the buffer when the remote peer acknowledges receipt. If there is no receipt acknowledged, the buffer rewinds its transmit pointer to the tail pointer and retransmits the entire window of unacknowledged data. Any error condition causes the packet to go unacknowledged, and the LLP retransmits all micropackets in the retransmit buffer until it receives a positive acknowledgment. The combination of the CCITT-CRC and the sliding-window protocol protects the link from all single, double, and odd number of bit errors, all burst errors up to 16 bits long, dropped and duplicate micropackets, and clock and data-framing glitches.

The LLP uses transparent link negotiation during reset to support multiple port widths. It can negotiate with chips having 10-, 20-bit, or wider interfaces. At reset, the peer LLPs negotiate using a narrow port width to arrive at the widest common port width. The interface to the chip core is identical in all bit modes, but the data rate reflects the port width. This is useful for interfacing to devices with varying data rate needs. In addition, if a Spider-to-Spider link is damaged in any of the upper 10 bits it can negotiate to use the lower 10 bits and operate at half bandwidth.

Message layer

The message layer defines four virtual channels[2] and provides independent flow control, header identification, and error notification on a micropacket basis. Flow control is credit based. After reset, all links credit their peers based on the size of their virtual-channel receive buffers. This size can vary between interface chips, but Spider implements 256-byte buffers for each virtual channel on each port. This size is sufficient to maintain full bandwidth over 5-meter cables and includes some extra buffering to absorb momentary congestion without degrading bandwidth.

The message layer transmits virtual channel tagging and

Credit	Er-	Tail	7	6	5	4	3	2	1	0
			Sideband bits							
Credit	No	No	0	0	Vcr 1	Vcr 0	Cr 1	Cr 0	Vch1	Vch0
		Yes	0	1	Vcr 1	Vcr 0	Cr 1	Cr 0	Vch1	Vch0
	Yes	No	1	0	0	Vcr 1	Vcr 0	Cr 0	Vch1	Vch0
		Yes	1	0	1	Vcr 1	Vcr 0	Cr 0	Vch1	Vch0
No credit	No	No	1	1	0	0	0	R	Vch1	Vch0
		Yes	1	1	0	0	1	R	Vch1	Vch0
	Yes	No	1	1	0	1	0	R	Vch1	Vch0
		Yes	1	1	0	1	1	R	Vch1	Vch0
Credit only			1	1	1	Vcr 1	Vcr 0	Cr 2	Cr 1	Cr 0

Vcr 1	Virtual channel no. to credit most significant bit
Vcr 0	Virtual channel no. to credit least significant bit
Cr 1	Credit value most significant bit
Vch 1	Virtual channel no. of this packet
R	Reserved

Figure 3. Message layer definition of sideband.

42	9	4	8	2
Data	Dest. ID	Dir.	Age	CC
Data				

Figure 4. Message layer header format. CC: congestion control.

credit information in the micropacket sideband field (Figure 3), leaving the remaining 128 bits for data transfer. The total overhead cost (in bandwidth) of providing reliable links, assigning virtual channels, and managing flow control is 32 bits out of 160—or 20%. This leaves the effective link bandwidth available for data transfer at 800 Mbytes/s per direction per link.

Message format

Each message contains a header micropacket (Figure 4) followed by zero or more data micropackets. Each header micropacket reserves 23 bits to specify destination and routing information. The header packet's remaining bits function as data and may be used in any way by a higher level protocol. A tail bit set on a message's last micropacket delineates the message boundary. Message headers always follow micropackets with the tail bit set.

A 9-bit destination identifier specifies one of 512 possible network destinations. These identifiers map into the routing tables, described later. The header direction field, which is 4 bits, specifies an exit port on the next Spider chip. This direction format supports up to 15 ports, as direction 0 is always reserved for communication with the Spider's local administration control block. There are 256 levels of message age, which the chip uses in arbitration. Finally, the header provides 2 bits of congestion control. The CC field's least-significant bit specifies that an individual message may adapt between two virtual channels, always choosing the least-used virtual channel at each port. This increases performance in heavily loaded networks, but allows messages to arrive out of order.

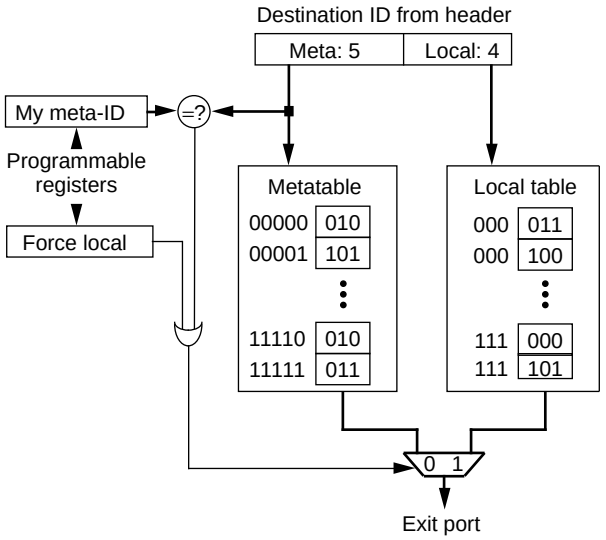

Figure 5. Programmable table format.

Routing

To route messages, the Spider chip uses programmable, distributed tables. Programming the tables establishes a static route between every two endpoints in the network. This allows hardware to make routing decisions in minimal time, while software is free to reconfigure the network to avoid faults or establish a new route map for performance or resource reasons. This scheme relieves the endpoints from storing global network knowledge and cumbersome source-routing hardware in the endpoint interface chips.

Each message header specifies a unique destination ID. When a message enters a port, the chip uses the destination ID to look up routing instructions in the tables. The table returns a direction, or exit port, which the next Spider chip uses for crossbar arbitration. We pipelined table lookup in this fashion to reduce latency.

We organized the routing tables into a hierarchy of two levels (Figure 5). This reduces the number of entries required in each table from 512 to 48, but also places some restrictions on routing possibilities. The destination ID's 4 least significant bits specify the local address, while the upper 5 bits specify the meta-address. Each Spider chip has a meta-ID register, which it compares to the message's meta-ID. If meta-IDs match, the message is in the correct metadomain, and the chip uses the local table to route the message to the precise destination. If meta-IDs do not match, the chip uses its metatable to specify the route to the correct metadomain.

It is possible to send messages without using the tables via a source-routing protocol called vector routing. Users employ this mechanism for network configuration and administration. Vector-routed packets contain a source-relative, step-by-step series of direction fields, which define a message's exact route. Users can send vector-routed messages while standard table-routed messages are in flight.

Spider's programmable tables support a variety of topologies. Different topologies trade off cost, network bandwidth,

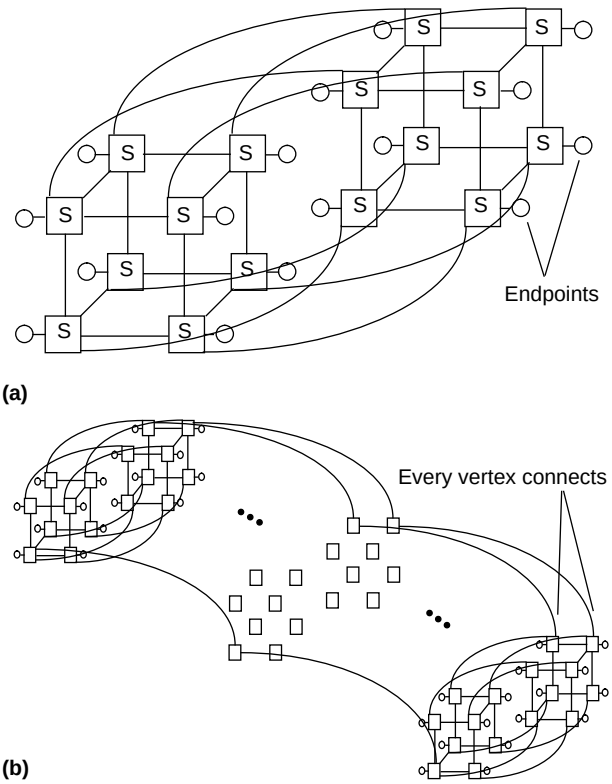

Figure 6. Hypercube topologies: 4D hypercube network with 16 endpoints (a) and 4D local, 1D fat metahypercube network with 64 endpoints (b).

and route redundancy. To avoid deadlock, tables must be programmed in a cycle-free manner. We describe two possible topologies here.

Hierarchical fat hypercube. To achieve scalable bisection bandwidth with minimal average distance between endpoints, users can choose a hierarchical fat hypercube as shown in Figure 6a (hypercube examples: a 0-dimension hypercube is a point; a 1D hypercube is a line; a 2D hypercube is a square; a 3D hypercube is a cube; a 4D hypercube is a tesseract). This topology grows as a standard hypercube for two to 32 endpoints, then expands to a series of metahypercubes for larger systems (Figure 6b). We call the metahypercubes fat hypercubes, because there is actually a full hypercube connecting each vertex of every local hypercube. The two levels of hypercube need not be of the same dimension.

The hierarchical fat hypercube topology maintains a constant bisection bandwidth of 800 Mbytes/s per endpoint for all configurations up to 512 endpoints. Networks can be sparsely populated, and the number of endpoints or Spider chips need not be powers of 2. This topology can sustain multiple faults in both the metahypercubes and the local hypercubes without losing connectivity.

To reduce cost, it is also possible to place two or more endpoints on each Spider chip, which is called bristling. Increasing a network's bristling factor will reduce the Spider chip count, but will also reduce network bisection bandwidth.

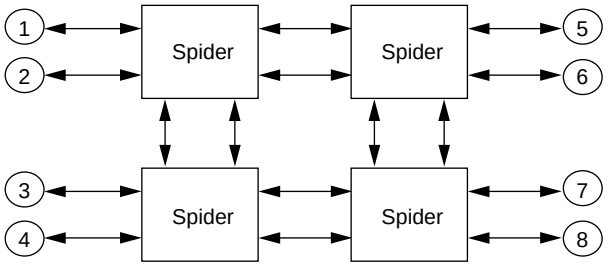

Figure 7. A nonblocking switch. Circled numbers represent endpoints.

Nonblocking *N*×*N* switch. Another possible network topology is a nonblocking *N*×*N* switch (see Figure 7). The topology uses O[(*N*×*N*)/4] Spider chips to build a full crossbar between *N* ports for large *N*. This topology has a guaranteed bandwidth of 1.6 Gbytes/s between every endpoint pair simultaneously, but is expensive for large configurations because it uses many Spider chips.

Crossbar arbitration

The central crossbar arbitration algorithm is crucial to performance. The goal is to maximize bandwidth through the crossbar while guaranteeing starvation avoidance and favoring higher priority messages. To minimize latency, when there is no output port contention, the arbiter falls into a bypass mode that uses fixed priority evaluation for minimal arbitration time.

To maximize bandwidth through the crossbar without using unreasonable buffering, we organized each virtual channel buffer as a set of linked lists. There is one linked list for each possible output port for each virtual channel. This solution avoids head-of-queue blockage, since a blocked message targeting one output port will not prevent a subsequent message in that same buffer from exiting on a different output port.

Using the linked-list-based virtual-channel buffers, the central arbiter attempts to maximize the number of assigned ports for each arbitration. This type of arbiter, with organization similar to that of the wavefront arbiter described by Tamir and Chi,[3] improves as the number of arbitration candidates increases. To maximize crossbar efficiency, each virtual channel from each port can request arbitration for every possible destination, providing up to 120 arbitration candidates each cycle.

To avoid starvation and encourage network fairness, the chip rotates the arbiter each arbitration cycle to favor the highest priority requester. Priority is based on the age field of a message header. Once a message enters a Spider chip, it ages at a programmable rate until it exits, taking its new age in the message header as it enters the next Spider.

This concept of network age promotes fairness as a function of network flight time rather than just across a single chip. It tends to reduce starvation problems brought on by network hot spots, because endpoints distant from the hot spot will have a higher priority when they arrive at the hot spot.

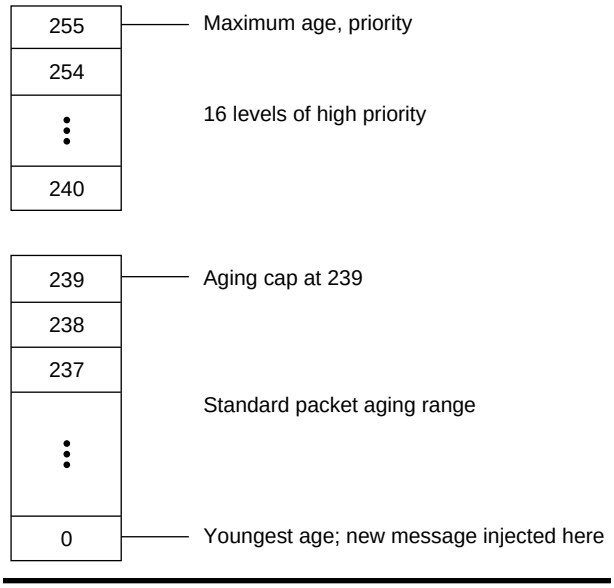

Figure 8. Message priority levels.

Users can inject messages considered high priority into the network with an artificially high age. We have reserved the top 16 age levels for the highest priority messages, as standard messages do not age beyond a value of 239 (see Figure 8).

Travel through the network may introduce gaps between data packets of the same message. This occurs when a higher priority message on a different virtual channel cuts through a lower priority message in progress or when port contention causes gaps due to buffer underflow. Users can program the Spider chip, however, to discourage such message gaps. Register settings allow a message above a threshold age to cut through other message in progress to minimize high-priority message latency. If users increase or disable the high-priority threshold for message cut-through, fewer gaps will appear in messages as they cross the network.

Network administration and error handling

The Spider chip core contains an administrative module that provides control functions and tracks error conditions and performance data. Registers in this module control functions such as aging rates, message cut-through thresholds, table access, error state, performance metrics, port protection, and port shutdown, as well as scratch registers with value exchange access. Protection registers control access to the administrative module, and users can set reset fences to prevent reset from propagating through network partitions.

Although the LLP layer ensures reliable transmission, it is important to track transient errors, because they affect performance and may indicate a failing part. The administrative module counts and stores checkbit errors and sequence numbers errors on a per port basis. If a chip retries a single packet several times without success, the administrative module shuts the link down. The system accepts no new data from the link until software examines the error state and resets the link.

Users can also set timers to watch for a tail time-out condition, which occurs when a message remains in progress over a certain virtual channel with no data flowing for a time-out period. When a tail time-out occurs, the Spider chip fabricates a tail micropacket, setting an error indicator, and sends it along the open channel. Another timer can check for deadlock conditions. If a given virtual channel remains full with no progress for a time-out period, the chip sets a deadlock time-out bit and resets the port. This condition should only occur due to misprogrammed tables or endpoint loops.

The Spider chip also provides a real-time clock network for applications that require tight synchronization between endpoints. Two additional physical wires per port distribute a clock signal throughout the network. Control registers select the source clock port, and all other endpoints receive the clock with minimal skew.

Physical design

The Spider chip is built in CMOS 5L, a 0.5-micron process from IBM. The 850,000 gates comprise a combination of standard cells, memory arrays, hand-laid data path macros, and custom I/O cells to handle high-speed off-chip signaling. The 160-mm^2 die with five metal layers connects to a custom 624-pin, 18-layer ceramic column grid array (CCGA) package using flip-chip C4-style die attach. The core operates at 3.3 V and dissipates up to 29 watts when all ports are in operation.

Three of the six ports provide a proprietary single-ended, open-drain, low-voltage interface. These ports can communicate with other chips in the same chassis and can tolerate multiple connector crossings. The other three ports drive complementary PECL output pairs. Differential signaling allows communication between chassis over cables up to 5 meters long, provided the ground shift between chassis is less than 500 mV. The differential ports also provide driver/receiver shutdown features that can function in concert with remote power sensing to provide hot plugging of cable connections. An external chip is also available to provide translation between single-ended and differential signaling levels.

Performance

One of the Spider chip's important applications is interprocessor communication, which is very sensitive to latency. To address this, we spent a great deal of design effort minimizing latency. The chips perform operations in parallel whenever possible and speculate arbitration before error status is known. We used custom cell layout to speed chip transit.

After data reaches the SSR and is synchronized, it enters the chip core and begins several operations in parallel. Table lookup and crossbar arbitration is normally serialized, as the exit port must be known before arbitration begins. To parallelize these operations, we pipelined table lookup across Spider chips. The direction field in the message header refers to the exit port targeted for the next hop, so crossbar arbitration can begin immediately. While arbitration progresses, so does table lookup for the next Spider chip, which depends on the destination ID and the direction field. This does

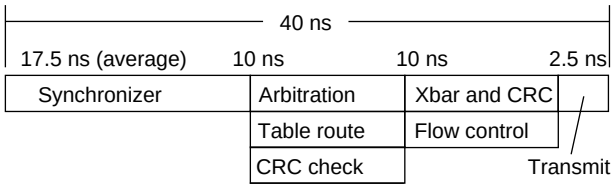

Figure 9. Pin-to-pin latency. The CRC needs to run twice per chip for each micropacket: once when the micropacket is received, once when the micropacket is sent.

increase table size, as a chip requires a full table for each neighboring Spider chip, but it reduces latency by a full clock cycle. Pipelined tables also add flexibility to possible routes, as the chips can assign different exit ports depending on where a message came from as well as where it is going.

During arbitration and table lookup, the LLP checks the CRC code and sequence numbers; it signals bad data two cycles later with a squash signal. When a squash occurs, all states affected by the squashed micropacket rewind. In the event that the chip has already routed the bad micropacket, the LLP appends a stomp code to the CRC on the outgoing micropacket.

If a message wins arbitration, it flows through the central crossbar and is joined by precomputed flow control information at the sender block. Finally, the LLP computes the CRC algorithm, and the SSD transmits the micropacket. The crossbar consists of hand-placed multiplexer cells, and we optimized the CRC generation by using wide parity cells.

In the absence of congestion, the Spider chip's pin-to-pin latency is 40 ns (see Figure 9). After adding interchip propagation delays such as circuit board trace and cables, uncongested delay is approximately 50 ns per Spider chip. Table 1 shows the average latency for uniform accesses between endpoints for the nonbristled, hierarchical, fat hypercube topology discussed earlier.

THE FIRST APPLICATION of the Spider chip is providing an interconnect for the SGI Origin product line. The Origin 2000 is a shared-memory multiprocessor that scales from two to hundreds of processors. Origin uses Spider to achieve systemwide cache miss latencies under 1 microsecond while scaling up linearly in bisection bandwidth. The Spider will also appear as the interconnect building block for the Stanford FLASH multiprocessor project, a distributed shared-memory system with a programmable protocol engine. The wide variety of programmable features incorporated in Spider will allow these systems to further study the benefit of each mechanism.

Table 1. Average latency.

No. of endpoints	Average latency (ns)	Bisection bandwidth (Gbytes/s)
8	118	6.4
16	156	12.8
64	274	51.2
256	344	205.0
512	371	410.0

Acknowledgments

A small but inspired team of SGI engineers made the Spider chip possible. We met all design and performance goals at speed and on time. Special thanks to the hardware team: Yuval Koren, Bob Newhall, and David Parry, with Ron Nikel on high-speed signaling. Thanks also to Dan Lenoski and Jim Laudon for architecture feedback, and to Dick Hessel and Yen-Wen Lu for performance simulation and feedback.

References

1. W. Stallings, *Data and Computer Communications,* Macmillan Publishing Co., Riverside, N.J., 1988, pp. 137-144.
2. W.J. Dally, "Virtual Channel Flow Control," *Proc. IEEE 17th Int'l Symp. Computer Architecture,* IEEE Computer Soc. Press, Los Alamitos, Calif., 1990, pp. 60-68.
3. Y. Tamir and H.-C. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches," *IEEE Trans. Parallel and Distributed Systems,* Vol. 4, No. 1, 1993, pp. 13-27.

Mike Galles is a principal engineer at Silicon Graphics Computer Systems, specializing in multiprocessor computer architectures. Since joining SGI, he has worked on the R4000 processor, the Challenge series bus-based symmetric multiprocessor (SMP), and, most recently, the Origin scalable shared-memory multiprocessor (S2MP).

Galles received a BSEE and an MSEE from Stanford University. He is a member of the IEEE Computer Society.

Direct questions concerning this article to Mike Galles, Silicon Graphics Computer Systems, 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311; galles@sgi.com.