SPIDER: Flexible and Efficient Communication Support for Point-to-Point Distributed Systems *

James Dolter, Stuart Daniel, Ashish Mehra, Jennifer Rexford,
Wu-chang Feng, and Kang Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

SPIDER is a network adapter that provides scalable communication support for point-to-point distributed systems. The device exports an efficient interface to the host processor, provides transparent support for dependable, time-constrained communication, and handles packet routing and switching. The communication support provided by SPIDER exploits concurrency between the independent data channels feeding the point-to-point network, and offers flexible and transparent hardware mechanisms. SPIDER allows the host to exercise fine-grain control over its operation, enabling the latter to monitor and influence data transmission and reception efficiently. In the current implementation, SPIDER interfaces to the Ironics IV-3207, a VMEbus-based 68040 card and will be controlled by x-kernel, a communication executive allowing the flexible composition of communication protocols.

1 Introduction

Traditionally, parallel computers and distributed systems have been employed in disparate application domains. Parallel computing has been motivated primarily by the need for high-performance scientific computing, resulting in regular interconnection networks and tightly-coupled processing elements. Distributed systems, on the other hand, arose from the need for connectivity, communication, and resource sharing between network-based machines. This paper presents SPIDER (*Scalable Point-to-point Interface DrivER*), a network adapter that combines the protocol support and media access of distributed systems with the low-level packet routing and switching schemes of the point-to-point, parallel computing domain.

In recent years distributed computing has emerged as a scalable and cost-effective solution to many classes of applications with widely-varying characteristics and resource requirements. Technological advances in VLSI, networking, and operating systems have expanded the domain of distributed computing, facilitating the merger of the seemingly disparate disciplines of parallel computing and distributed computing. Faster networks now allow distributed systems to employ mechanisms previously applied only to tightly-coupled parallel machines, including system-wide shared memory and a finer grain of computation. In addition, parallel programming abstractions are now being applied across a wide variety of distributed computing platforms.

It is also becoming commonplace to use digital computers for real-time applications such as fly-by-wire, industrial process control, computer-integrated manufacturing, and medical life-support systems. These applications impose stringent timing *and* dependability requirements on the computer system, since a disruption of service caused by a physical failure or inadequate response time can result in a catastrophe. Commonly, dependability is provided by incorporating some form of redundancy into the system. One technique replicates critical software components on a collection of nodes that fail independently [5]. Coordinating this software replication necessitates timely and dependable communication between nodes.

Point-to-point networks, with their multiplicity of processors and internode routes, provide a natural platform for applications that require both high performance and dependability [18]. Many parallel computers connect the processing elements with a point-to-point network [7, 9, 11, 20] to provide scalable communication bandwidth to applications. However, these networks often consist of short links, such as on-board wires or ribbon cables, with no need for higher-level error control. Centralized hardware and software can make parallel machines vulnerable to single-point failures. For example, the message-driven processor (MDP) [9] for the J-machine is a chip that connects to a 3D-mesh network. With 64 nodes on a board and multiple boards in a chassis, a single board failure can disrupt several processing elements.

*The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the NSF.

A network of physically-distributed computers offers the advantage of independent processor and link failures. Each node can erect firewalls against failures by employing protection boundaries. Communication protocols, tailored to the delay and error characteristics of the network media, mediate between communicating entities. However, in many distributed systems nodes interface to the communication fabric through only one or two ports [10, 16, 24]. In this configuration, one or two media failures can partition the network. For example, the Nectar communication accelerator board (CAB) [24] for heterogeneous distributed computing supports one bidirectional port into the network, and thus, a single failure can isolate a node.

Combining the high connectivity of point-to-point, parallel machines with the communication protocols of distributed systems results in a hybrid environment well suited to both real-time and non-real-time applications. SPIDER is a front-end hardware module designed to provide communication in this composite domain. SPIDER supports protocol processing through a low overhead interface to the host processor, while handling low-level routing and switching and providing transparent hardware support for dependable, time-constrained communication.

The following section presents a brief design overview of SPIDER. Section 3 describes the device's interface to the controlling host processor, while Section 4 discusses SPIDER's flexible control close to the physical links. Section 5 describes how the internal operation of the device supports predictable, dependable communication between nodes. The paper concludes by presenting our current hardware and software platform and outlining our short-term and long-term goals.

2 Design Overview

The boundary between hardware and software in the communication subsystem determines the functionality, and hence the complexity, of the adapter. Many intelligent, complex adapters reduce the load on the host significantly, but sacrifice flexibility regarding communication protocols and buffer management strategies. Instead of implementing a specific protocol in VLSI [6], SPIDER provides hardware support without precluding higher-level host policies.

While this flexibility could be achieved through a pure software implementation, hardware support provides several advantages. In point-to-point systems with a high degree of connectivity, servicing the multiple incoming and outgoing channels entirely in software imposes substantial overhead; this alone could overload a conventional processor during peak loads. Instead, SPIDER pushes software control as close to the links as possible by dedicating a small processor to each channel.

2.1 Architecture

Designed to reside on the memory bus of the host processor, SPIDER has direct access to the host memory and provides the host processor with memory-mapped access to the control interface. User applications may run on this host or on another processor that uses the host, coupled with SPIDER, as a dedicated communication engine. While the host is responsible for the presentation, session, and transport layers, SPIDER handles the data link and physical layers, as well as part of the network layer. A collection of SPIDERs can form point-to-point networks with a variety of topologies, including rings, meshes, and irregular configurations.

Much of the design of SPIDER centers around providing fair and efficient coordination of the multiple, independent data channels. As shown in Figure 1, SPIDER manages bidirectional communication with up to six neighboring nodes, with two virtual channels on each unidirectional link. The programmable routing controller (PRC), a 187-pin custom integrated circuit designed using the Epoch silicon compiler, is the main component of SPIDER [12]. The 12 PRC TXs provide low-level control of packet transmission while the 12 microprogrammable PRC RXs coordinate packet reception, as well as low-level routing and switching. The PRC TXs and PRC RXs serve as small, custom processors that implement the low-level drivers controlling the actual transmitter and receiver devices on the network interface (NI). To maximize flexibility, each PRC RX has a 128-word control store to which the host downloads microcode during system initialization.

The PRC devices reserve access to outgoing channels (NI TXs) via an on-chip reservation status unit. The NI performs the media access control on six pairs of AMD TAXI chips [2], where each TAXI TX/RX pair provides bidirectional communication with an adjacent node. A TAXI transmitter accepts injected data from two PRC TXs and each TAXI receiver delivers data to two PRC RXs, providing two virtual channels for each physical, unidirectional link. The NI TX and NI RX control units perform the necessary interleaving of channels to and from the physical link, on a byte-by-byte basis.

The host can influence operation in the PRC RX through notification FIFOs, addressable as part of SPIDER's memory-mapped control interface. These FIFOs provide bidirectional information exchange between a PRC RX and the host. The host controls channel reservations for any packet stored in the host memory by assigning the outgoing packet to a particular PRC TX. The host transmits a packet by feeding this PRC TX with page tags, each of which includes the address of an outgoing page and the number of words on the page. Likewise, the host provides each PRC RX with pointers to free pages in the memory, for use by arriving packets. The control interface also provides read access to an event queue that logs page-level activities in the PRC.

2.2 Basic Operation

To illustrate the interaction between the host, SPIDER, and the network, consider how a message is handled as it travels from the source node, through an intermediate node, and to the destination node.

Transmission: When an application requests that the host transmit a message to another node, the host disassembles the message into one or more packets,

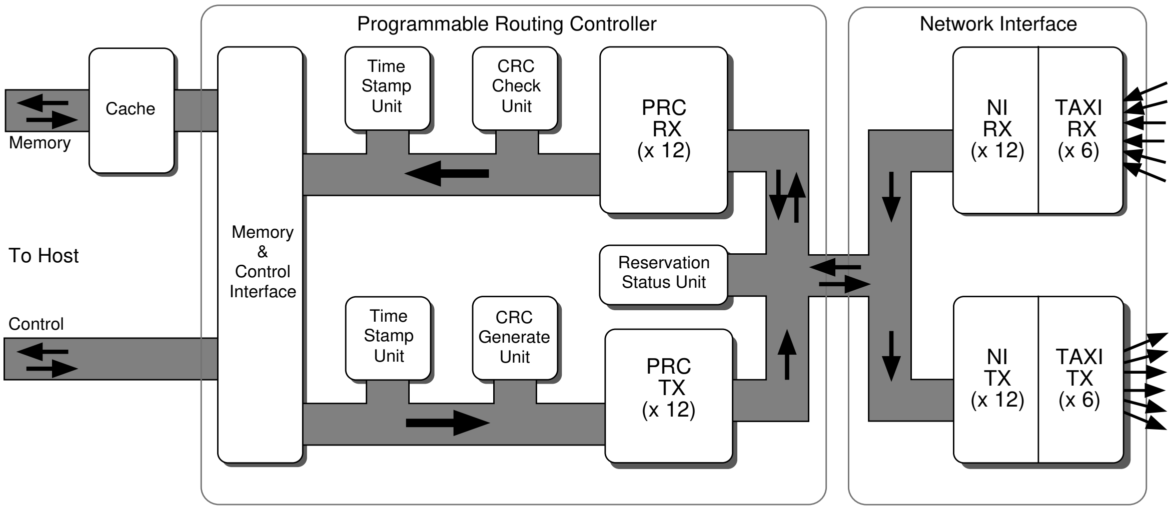

Figure 1: SPIDER architecture

where each packet consists of one or more (possibly non-contiguous) 64-word pages. Using the control interface, the host then instructs the appropriate PRC TX to transmit these pages. The PRC TX uses the memory interface to fetch the data on each page. Data is initially fetched from the host memory in cache lines and converted to words as the packet travels from the host memory interface to the PRC TX. During the data transfer from the memory interface to the PRC TX, the PRC transparently accumulates a 32-bit cyclic redundancy code (CRC). After sending the last data word of the packet, the PRC TX transmits a 32-bit timestamp, read from a counter on the PRC, followed by the CRC. The PRC TX transmits each of these words to the NI a byte at a time.

Cut-through: Packet reception begins when data arrives in a TAXI RX in the network interface (NI). The NI RX initially forwards data to its associated PRC RX, until the PRC RX has received enough header bytes to make a routing decision for the packet. If the packet is destined for a subsequent node, the PRC RX can try to send the packet directly to the next node by reserving an NI TX. If the PRC RX is able to establish a cut-through [7, 19], the PRC RX then sends the data it has accumulated to that transmitter and reconfigures the NI RX to forward data directly to the reserved NI TX, bypassing the PRC entirely. When the packet has cleared the node, the NI RX automatically reconfigures itself to forward the next packet to its associated PRC RX.

Reception/Buffering: When a packet buffers at the local node, however, the PRC RX simply collects the bytes from the NI, reaccumulating the CRC while transferring each word of the packet to the memory interface. These words are assembled into cache lines and written into pages in the host memory. The PRC event queue logs the arrival of each page, noting the address and size. At the end of the final page of the packet, the PRC RX appends a receive timestamp to the packet and logs a packet-arrival event indicating the outcome of the CRC check. If the packet has reached its destination, the host reassembles the pages into a packet and the packets into a message. Otherwise, the host schedules the packet for transmission to the subsequent node in its route.

3 Host Interface

The management of several, concurrently active, data channels has implications for protocol processing, buffer management, packet scheduling, and interrupt handling. The host must be able to handle a higher frequency of events, maintain distinct scheduling queues for each channel, and service each channel in a fair and efficient manner. Although it is possible to construct a point-to-point distributed system using replicated communication hardware at each node, cost and performance considerations necessitate an integrated, low overhead design for the communication adapter.

While SPIDER could provide separate memory and control ports for each channel, this would require replicating bus arbitration logic for each channel, resulting in more expensive hardware and increased bus loading. Instead, the PRC multiplexes the 24 independent channels, allowing SPIDER to provide a single, shared port to the host. This interleaving of channels significantly reduces the pin-out of the PRC, allowing an integrated, single-chip solution.

3.1 Packet Transfer

Information structures facilitating efficient host-SPIDER coordination, such as the event queue and the page tag queues, have been moved into SPIDER. SPIDER imposes few restrictions on packet format or size since it sees a packet only as a collection of one or more pages. This minimizes data copying overhead and host memory fragmentation. Consistency of packet data between SPIDER and the host is ensured by holding the host bus when transferring a cache line and trusting the host not to alter pages that it currently has allocated to SPIDER. These pages can be partially filled and need not be contiguous in memory, similar to MBUFs and MBUF clusters [22].

To transmit a packet, the host creates the header on a separate page, leaving the data pages untouched, and feeds the page tags to the page tag queue for the PRC TX selected by the routing algorithm. Generating the header on a separate page avoids unnecessary data copying, thus keeping the overhead of network data transfer low [4, 10]. The first page tag for a packet can specify a number of words to be excluded from the CRC calculation; subsequent modification of these words by other nodes does not invalidate the original CRC checksum. As SPIDER fetches and transmits individual pages, it logs events in the event queue indicating the transmission of each page and of the entire packet. The host can respond to these events by providing the PRC TXs with additional page tags or updating state information for this packet.

In order to receive a packet, the host fills addresses of free pages in host memory into a single page tag queue shared by all PRC RXs. As individual pages are received and placed in host memory, SPIDER logs events corresponding to the reception of each page and of the entire packet. The host responds to these events by providing additional free pages for the PRC RXs and updating the appropriate state information. The event queue contains the information necessary for the host to reconstruct the packet from the arriving pages.

3.2 Page-Level Flow Control

SPIDER enables the host to employ different packet scheduling algorithms, and use arbitrary data structures for scheduling messages and packets. The page tag queues for outgoing channels are fed from these host-managed scheduling queues. The command set exported by SPIDER allows the host to directly influence data transfer on each of the active data channels. The host can efficiently assess the status of all the page tag queues simultaneously. In addition, the event queue logs SPIDER's use of pages for both transmission and reception, providing necessary feedback to the host to influence packet scheduling and free-list maintenance. Using this feedback the host can exercise page-level control over the rate at which data is fed into these channels.

An arriving packet can cut through an intermediate node if the next link in the route is free, thus avoiding buffering and subsequent processing at that node. If busy outgoing channels prevent an immediate cut-through, SPIDER may begin buffering the packet. If the link becomes free during this buffering, the host can reschedule the partially-arrived packet for transmission. Such partial cut-throughs [1, 19] can potentially improve performance for large packets by overlapping the forwarding of buffered pages with the arrival of subsequent pages of the packet. SPIDER facilitates partial cut-throughs by maintaining page-level information in the event and page tag queues.

With several data channels to service, the host could easily be overwhelmed by interrupts alone. SPIDER incorporates mechanisms to minimize the number and frequency of interrupts delivered to the host. Further, the host can amortize the cost of fielding an interrupt by reading the entire event queue during each interrupt or polling cycle. The host can parse the events registered in the event queue into several internal event queues, possibly corresponding to different priorities. These events can be handled at a later time, enabling the host to keep the interrupt service routines short. Interrupt masking can be employed to disable non-essential interrupts if polling SPIDER would be more cost-effective. Coupled with parsing the entire SPIDER event queue into internal event queues, this also reduces the number of context switches caused by interrupts.

4 Routing and Switching

While the host manages communication at the page level, SPIDER coordinates the fine-grain interaction between incoming and outgoing channels. The virtual channel abstraction transcends the multiple protocol layers in the device, allowing host operations at the control interface to influence routing and switching operations at the links. SPIDER's flexibility enables the host to dynamically tailor routing and switching schemes according to prevailing network conditions and communication requirements.

4.1 Network Interface

Communication in point-to-point networks requires multiplexing incoming traffic to the multiple outgoing links. Many routers for point-to-point, parallel machines employ some type of crossbar interconnect [7, 20, 28]. As a result, multicomputer routing chips are often pin-limited [1]. Instead, SPIDER uses a demand-slotted, time-division multiplexed bus to connect the incoming and outgoing channels. The demand-slotted bus prevents idle channels from interfering with active devices. This facility can prove helpful in dissipating congestion and traffic hot-spots by maximizing bus utilization. Fair arbitration prevents any one channel from stalling packets on other channels.

The SPIDER network bus allows a single transaction to spawn transmissions on several links simultaneously. When forwarding data, a master device on the bus can specify one or more NI TXs as slaves, providing multicast support on the byte level. Using this facility, an arriving packet can spawn multiple copies at each hop on its route, allowing SPIDER to support efficient broadcast and multicast algorithms [17]. The host processor can utilize this multicast facility to support efficient group communication, useful for establishing consensus amongst cooperating software on different nodes.

The bus interface defines a simple protocol for addressing and accessing the NI devices. Effectively, the NI is a plug-replaceable module that handles low-level flow control and the multiplexing of virtual channels to physical links. Insulating the rest of SPIDER from the low-level media access control [10] allows SPIDER to interface to various communication fabrics. Other NI designs can change the granularity of network flow control or the interleaving of PRC channels to the physical links. With different multiplexing control, each PRC can support up to twelve neighbors with a single virtual channel per link, up to six neighbors with a pair of virtual channels per link, or other combinations. These virtual channels can be used for deadlock-free wormhole routing [8, 25] or to partition various types of traffic onto different virtual networks.

4.2 Microprogrammability

Routers for parallel point-to-point networks typically employ a single routing and switching scheme, implemented in hardware. For example, the TRC [7] and the MDP [9] perform e-cube routing with wormhole switching, while the Chaos router [20] uses a type of deflection routing with virtual cut-through switching. Earlier systems often implemented packet switching with some form of static routing. The various routing and switching schemes have different characteristics, in terms of latency, deadlock-avoidance, and predictability.

The microprogrammable PRC RXs allow SPIDER to download algorithms for a wide variety of routing and switching combinations. Based on the header of an arriving packet, a PRC RX can select the next node in the packet's route and choose whether to buffer, stall, forward, or drop the packet. The microprogrammable PRC RXs allow the PRC to support circuit switching, packet switching, virtual cut-through switching [19], and wormhole switching [7]. The notification FIFOs provide a rendezvous point between each PRC RX and the host, and are used to diagnose and respond to dynamic conditions, such as congestion or faulty links. This enables the host to tailor routing and switching policies to application requirements and the state of the network. With different microcode, SPIDER can implement various routing algorithms [13], including adaptive and nonminimal schemes.

Since each PRC RX has its own microcode control store, each incoming virtual channel can impose its own set of routing and switching policies. For example, real-time messages generally use packet switching and static routing for predictable performance [18], although best-effort packets can improve their average latency by using cut-through switching and adaptive routing. Carrying these two types of traffic on different virtual channels allows real-time communication to coexist with non-real-time packets without sacrificing the performance of either class.

Programmable routing allows SPIDER to handle irregular network topologies. Parallel computers typically have a regular interconnection topology, conducive to scientific computation. However, distributed systems often operate with irregular topologies, such as wide-area networks or distributed control systems. A regular mesh network may not be suitable for a system coordinating manufacturing on a factory floor. Even when a regular topology is appropriate, broken communication links can render the system temporarily inoperable if the router cannot avoid the down links. The microcoded algorithms in the PRC RXs enable arriving packets to circumvent faulty links and nodes.

5 Time-Constrained Communication

The design of SPIDER emphasizes support for dependability and predictability without tying the host to a particular strategy. To make a system dependable, some form of redundancy is usually incorporated into the system. This redundancy can be achieved by replicating critical software on groups of nodes [5], but this requires the provision of timely and dependable communication. To support software replication, SPIDER supports efficient group communication through its multicast facility, and fault-tolerant routing through the microprogrammable PRC RXs. In addition, SPIDER provides transparent error detection and bounded communication delays.

5.1 Dependability

Providing dependability in a system with unreliable communication media requires some form of error control. In many cases, packet type dictates the error control strategy. Certain high-priority messages with short deadlines might need to mask errors entirely, while periodic sensor readings could be discarded if found to be erroneous. SPIDER's design, therefore, provides efficient error detection while relegating error recovery to the higher-level protocols.

Error detection is usually provided through the calculation and transmission of a CRC with each packet [3]. Software CRC calculations provide the highest degree of flexibility and also detect errors introduced during the transfer of packets between the network interface and main memory. This calculation, however, requires host access to the entire packet on both transmission and reception. Allowing the network interface hardware to calculate the CRC can virtually eliminate this cost, as the calculation can be made by the hardware during the transfer of the packet to and from the network interface. This is the approach taken by the Nectar CAB [24] and Afterburner [10]. But whereas these designs manage a single bidirectional port into the network, SPIDER must handle twelve.

To avoid replicating hardware, the PRC uses a single 32-bit parallel CRC generator to compute the CRC for every outbound packet. A single checker accumulates the CRC for incoming packets en route to the memory interface. The outcome of the CRC check is logged in the PRC event queue as part of the event indicating packet reception. Since many routing algorithms may wish to modify the header of a packet in the PRC RX, SPIDER allows a portion of the packet to be excluded from CRC calculations. The microprogrammable PRC RXs can then enforce additional error detection or correction on the routing header.

SPIDER's error detection allows the system to diagnose faulty links by logging CRC errors on channels. After higher-level fault diagnosis, individual nodes can propagate this information to their PRC RXs through the notification FIFOs, allowing future packets to circumvent the faulty link.

5.2 Predictability

Real-time applications mandate predictable system operation. Higher-level protocols for predictable packet scheduling depend on a global time base and bounded communication delays [18]. A global time base is necessary for any deadline-based scheduling, as well as many consensus protocols. While many parallel machines distribute a single clock to the processing elements [9, 23], the potential distance separating the nodes of a distributed system causes clock skews that make this approach infeasible. Distributed systems, therefore, must have some means of keeping the local clock of each node within some allowable skew. This can be done in software, hardware, or a combination of both [26].

Hybrid algorithms using a combination of hardware and software can keep the local clocks of a distributed system synchronized, while using simpler hardware and providing tighter bounds than software algorithms. SPIDER provides hardware support for such algorithms by timestamping each packet upon transmission and reception, using timestamp registers that are readable and writable by the host. By affixing timestamps close to the physical links, SPIDER provides an extremely accurate measure of when an outgoing packet completes injection and when an incoming packet completes reception at a node. This allows the host to provide tighter bounds on clock skews between nodes.

A distributed system cannot be predictable without some bound on communication delays. Therefore, SPIDER is designed to bound low-level communication delays. Access to interfaces in SPIDER is governed by fair, demand-slotted arbitration [21], and all of the interfaces are designed to avoid blocking. This allows SPIDER to guarantee that all channels can access the memory within certain bounds. Although the bandwidth provided to a particular channel can vary dynamically with the load, the worst-case is known and deterministically bounded.

6 Conclusion

SPIDER manages several concurrent data channels in a fair and efficient manner, while minimizing interaction with the host. The PRC TXs and PRC RXs serve as small, dedicated processors implementing low-level drivers controlling the network interface. The microprogrammable PRC RXs allow SPIDER to provide this fine-grain control without tying the host to a single routing and switching scheme. SPIDER's flexible design enables experimentation with a variety of existing and future communication protocols in point-to-point distributed systems.

The PRC has been designed in a 0.8μm process using Cascade Design Automation's Epoch Silicon Compiler and functionally simulated using Verilog-XL. The chip currently has 187 pins and measures 1.30 by 1.34 centimeters. It operates asynchronously on the memory and control interfaces, while the NI runs at 25 MHz. The SPIDER design interfaces to the Ironics IV-3207 [15], a VMEbus-based 68040 card, through a daughterboard interface to the processor memory bus. An IV-3207 card, coupled with the SPIDER daughterboard, can serve as a network-based uniprocessor handling both application and communication tasks or as a dedicated communication processor for a multiprocessor.

SPIDER will initially provide communication support for the Hexagonal Architecture for Real-Time Systems (HARTS) [27], a point-to-point, distributed system targeted for real-time applications. pSOS+ [29], a commercial real-time executive, provides system support to application threads within a node while x-kernel [14] coordinates communication between nodes. We are extending x-kernel to support point-to-point communication and real-time protocols using SPIDER. This platform allows investigation of software paradigms for managing time-constrained communication in point-to-point, distributed systems.

Our goal in designing SPIDER was to provide support for predictable, dependable communication while retaining the flexibility to experiment with a variety of existing and future communication protocols over several network topologies and under a variety of traffic characteristics. With SPIDER, we demonstrate that this goal can be met through a low overhead, integrated solution which supports, but does not dictate, higher-level host policies.

References

[1] S. Abraham and K. Padmanabhan, "Constraint based evaluation of multicomputer networks," in *International Conference on Parallel Processing*, pp. I–521–I–525, 1990.

[2] *Am79168/Am79169 TAXI^{tm}-275 Technical Manual*, Advanced Micro Devices, ban-0.1m-1/93/0 17490a edition.

[3] G. Albertengo and R. Sisto, "Parallel CRC generation," *IEEE Micro*, pp. 63–71, October 1990.

[4] D. Banks and M. Prudence, "A high-performance network architecture for a PA-RISC workstation," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 191–202, February 1993.

[5] M. Chereque, D. Powell, P. Reynier, and J. Voiron, "Active replication in Delta-4," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 28–37, 1992.

[6] G. Chesson, "XTP/PE overview," in *Conference on Local Computer Networks*, October 1988.

[7] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[8] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[9] W. J. Dally et al., "Design and implementation of the message-driven processor," in *Proc. Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, pp. 5–25, 1992.

[10] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner," *IEEE Network Magazine*, pp. 36–43, July 1993.

[11] A. L. Davis, "Mayfly: A general-purpose, scalable, parallel processing architecture," *Lisp and Symbolic Computation*, vol. 5, no. 1/2, pp. 7–47, May 1992.

[12] J. Dolter, *A Programmable Routing Controller Supporting Multi-mode Routing and Switching in Distributed Real-Time Systems*, PhD thesis, University of Michigan, September 1993.

[13] S. Felperin, L. Gravano, G. Pirarre, and J. Sanz, "Routing techniques for massively parallel communication," *Proceedings of the IEEE*, vol. 79, no. 4, pp. 488–503, April 1991.

[14] N. C. Hutchinson and L. L. Peterson, "The x-Kernel: An architecture for implementing network protocols," *IEEE Trans. Software Engineering*, vol. 17, no. 1, pp. 1–13, January 1991.

[15] *IV-3207 VMEbus Single Board Computer and Multiprocessing Engine User's Manual*, Ironics Incorporated, 1991 edition.

[16] H. Kanakia and D. R. Cheriton, "The VMP network adapter board (NAB): high-performance network communication for multiprocessors," *Proceedings of the SIGCOMM Symposium*, pp. 175–187, August 1988.

[17] D. D. Kandlur and K. G. Shin, "Reliable broadcast algorithms for HARTS," *ACM Trans. Computer Systems*, vol. 9, no. 4, pp. 374–398, November 1991.

[18] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 300–307, May 1991.

[19] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, no. 4, pp. 267–286, September 1979.

[20] S. Konstantinidou and L. Snyder, "Chaos router: Architecture and performance," in *Proc. Int'l Symposium on Computer Architecture*, pp. 212–221, May 1991.

[21] A. Kovaleski, S. Ratheal, and F. Lombardi, "An architecture and interconnection scheme for time-sliced buses in real-time processing," *Proc. Real-Time Systems Symposium*, pp. 20–27, 1986.

[22] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.3BSD Unix Operating System*, Addison Wesley, May 1989.

[23] C. Leiserson, Z. Abuhamdeh, D. Douglas, C. Feynman, M. Ganmukhi, J. Hill, W. D. Hillis, B. Kuszmaul, M. St. Pierre, D. Wells, M. Wong, S.-W. Yang, and R. Zak, "The network architecture of the connection machine CM-5," in *Symposium on Parallel Algorithms and Architectures*, pp. 272–285, June 1992.

[24] O. Menzilcioglu and S. Schlick, "Nectar CAB: A high-speed network processor," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 508–515, May 1991.

[25] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, pp. 62–76, February 1993.

[26] P. Ramanathan, K. G. Shin, and R. W. Butler, "Fault-tolerant clock synchronization in distributed systems," *IEEE Computer*, pp. 33–42, October 1990.

[27] K. G. Shin, "HARTS: A distributed real-time architecture," *IEEE Computer*, vol. 24, no. 5, pp. 25–35, May 1991.

[28] Y. Tamir and G. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 725–737, June 1992.

[29] L. M. Thompson, "Using pSOS$^+$ for embedded real-time computing," in *Proc. COMPCON*, pp. 282–288, 1990.