LIMITED DISTRIBUTION NOTICE
(NOT TO BE REMOVED FROM DOCUMENT):

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communication and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

Architecture and Implementation of Vulcan

Craig B. Stunkel Dennis G. Shea Bülent Abali
Monty M. Denneau Peter H. Hochschild Douglas J. Joseph
Ben J. Nathanson Michael Tsao Philip R. Varker

IBM Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Abstract

IBM's recently announced Scalable POWERparallel[1] family of systems is based upon the Vulcan architecture, and the currently available 9076 SP1[1] parallel system utilizes fundamental Vulcan technology. The experimental Vulcan parallel processor is designed to scale to many thousands of microprocessor-based nodes. To support a machine of this size, the nodes and network incorporate a number of unusual features to scale aggregate bandwidth, enhance reliability, diagnose faults, and simplify cabling. The multistage Vulcan network is a unified data and service network driven by a single oscillator. The attempt is made to detect all network errors via CRC checking and component shadowing. Switching elements contain a dynamically allocated shared buffer for storing blocked packet flits from any input port. This paper describes key elements of Vulcan's hardware architecture and implementation details of the Vulcan prototype.

[1] A trademark of International Business Machines

1 Introduction

The experimental Vulcan parallel processor is designed to scale to many thousands of microprocessor-based nodes. To support large, reliable systems, Vulcan incorporates a number of unusual features. The Vulcan network is a unified data and service network driven by a single oscillator. Network devices communicate synchronously [1, 2, 3] but need not be clocked by the same phase of the global oscillator. Instead, synchronization is accomplished via phase adjustment of data signals [2]. Each switching element contains a large shared buffer that is dynamically allocated, dependent upon requests from the element's input ports. The communication protocol maintains maximum bandwidth even for long communication links.

Vulcan was designed—and a working prototype built—in the Parallel Systems department at IBM Research. This paper describes the hardware architecture and implementation of the Vulcan machine. An equally ambitious scalable parallel software effort was undertaken in the same department.

The Vulcan prototype machine illustrated in Figure 1 is a small system that utilizes two large ASIC chips: the Vulcan Switch chip, and the Memory & Switch Management Unit (MSMU). The 8×8 Switch chip is the basic switching element of the network, while the MSMU functions as the interface between a Switch chip and a processing node, and optionally serves as a memory controller. Three flavors of nodes incorporating the MSMU have been implemented: processor nodes, I/O nodes, and host nodes. The Switch chip is incorporated into two basic network building blocks: (1) a small switch board with 8 bidirectional connections, and (2) a large 16-way to 16-way bidirectional board that forms a two a large 16-way to 16-way bidirectional board that forms a two stage building block for bidirectional multistage networks. The prototype was completely functional in early 1992.

IBM's recently announced Scalable POWERparallel Systems family, including the 9076 SP1, is based upon the Vulcan architecture. For example, the SP1 network is composed of components built for the Vulcan prototype, including the Switch and MSMU chips, the 16-way switch board, and the communication links. The Vulcan network service software is incorporated in the SP1 service strategy. Additional Vulcan prototype features will appear in future SP family releases.

Basic Vulcan network architecture is outlined in §2. The Vulcan switch chip is detailed in §3. Details of the Vulcan prototype are found in §4. §5 overviews the software written to service, support, and initialize the network. In §6 we examine some of the lessons learned from the prototype.

2 Architecture

The most important factor influencing the Vulcan architecture is the design point of a teraflop machine connected by a terabyte per second network. With current microprocessor technology, this design point requires a system with several thousand processors. A network

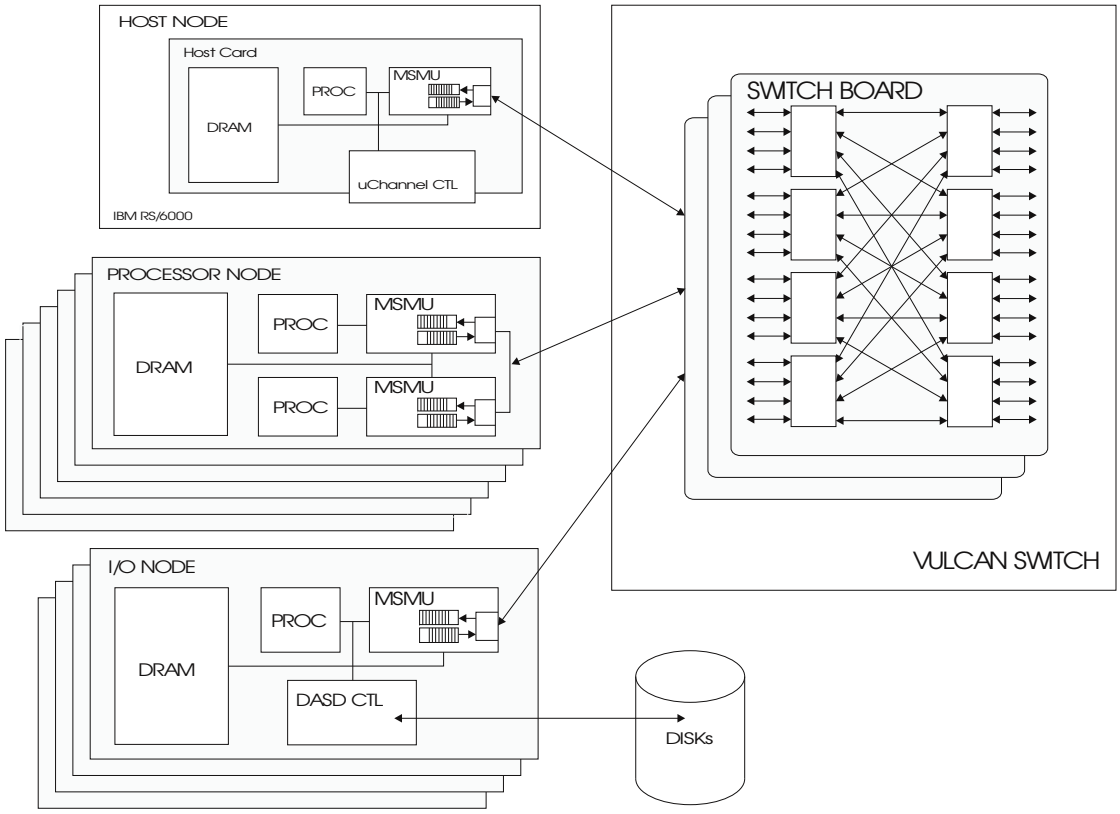

Figure 1: The Vulcan prototype system

capable of scaling to this size necessarily scales its aggregate bandwidth linearly with respect to the number of nodes. The network must provide low latency and be fault-tolerant, serviceable, and reasonably low-cost. A related consideration is efficient and fair access to I/O facilities attached to the network.

An orthogonal design consideration is modularity. We attempted to design a network that, although upwardly scalable, is efficiently implemented for only a few nodes as well. A final design criteria—related to scalability—is to leverage the latest in microprocessor technology. This presumably provides higher performance per unit cost.

2.1 Preliminary definitions

A network comprises two device types—*switching elements* and *nodes*—interconnected by communication links. Nodes are non-switching devices such as processors, I/O nodes, and gateways. Switching elements contain a plurality of input ports and a plurality of output

ports, and provide means of passing data arriving at an input port to an appropriate output port.

Nodes send messages to other nodes by breaking messages into *packets* and injecting these packets into the network. Each packet contains route information examined by switching elements to forward the packet correctly to its destination.

The smallest unit on which flow-control is performed is called a flow control digit, or *flit* [4]. In the Vulcan prototype a flit is one byte. The width of the data transmitted by an output port is also one byte.

A common measure of aggregate network bandwidth is *bisection bandwidth*. Here, bisection bandwidth is defined as the maximum achievable bandwidth across a worst possible cut in the network, where this cut divides the system into halves with identical numbers of nodes.

2.2 Network topology

We chose a bidirectional multistage interconnection network (MIN) topology. Common MIN's such as Omega [5], Banyan [6], and indirect binary n-cube [7] networks provide bisection bandwidth that scales linearly with the number of nodes while maintaining a fixed number of communication ports per switching element. Scalable networks are richly connected and require some long communication links to support large numbers of nodes. Thus, a scalable network should employ a protocol that maintains bandwidth over long links (see §2.5.1).

In MIN's nodes are essentially equidistant. The scheduler's assignment problem is thus simplified. In fact, random process assignment is often beneficial in reducing hot-spot contention. Shortest path routing algorithms for MIN's are typically free of deadlock. For MIN's that incorporate redundant paths, this property provides greater latitude in route selection.

In a bidirectional MIN [8, 9] each communication link comprises two channels, which carry data in opposite directions. The switch board in Figure 1 illustrates two stages of a bidirectional MIN constructed from 4-way to 4-way bidirectional switch elements. Each switching element can forward packets from any input port to any output port (including ports on the same "side"). Vulcan switching elements—physically 8 input, 8 output devices—are wired as bidirectional elements. Bidirectionality facilitates modular systems—it is simpler to build systems in a wide range of sizes when each system is not required to wrap links from the final stage's output ports back to the node input ports (at differing stages dependent upon system size). Bidirectionality also naturally incorporates redundancy. Bidirectional links also simplify our service software (§5.1) by allowing the software to fully test a single link or device while relying solely on proven intermediate devices. Bidirectional MIN's are related to practical implementations of fat-tree networks [1] derived from Leiserson's idealized fat-trees [10]. The Vulcan network topology resembles bidirectional SW Banyan networks [8], which are topologically equivalent to practical fat-trees for

which the number of parents at every intermediate stage is equal to the number of children. For this case, the tree never narrows, and bisection bandwidth scales linearly.

2.3 Single global oscillator

The Vulcan network operates synchronously—all network devices are driven by an image of a single oscillator. It is not necessary to adjust the phase of each device's oscillator image. Instead, each device adjusts the phase of *data* by a process we call *tuning* (see §5.1), using adjustable delay elements. This allows both the oscillator signal links and the data links to be of arbitrary length up to limits posed by reliability or protocol considerations.

Synchronous operation provides several indirect benefits in Vulcan:

- Packet error checking is conducted on a time basis, rather than a packet basis (see §2.5.1). Stuck or open faults are detected in a timely fashion.
- A Vulcan network can, if desired, operate as a network slice analogous to a bit-sliced ALU. This requires the ability to "tune" the corresponding links of each slice in the same manner.
- Synchronous operation, in conjunction with appropriate hardware mechanisms, facilitates the achievement of a closely synchronized and non-drifting *global time*, trivializing the well-known clock synchronization problem [11]. Each device in the Vulcan network accesses this time via its local copy: a loadable incrementer called the Absolute Time Counter (ATC). During the initialization of a Vulcan network, the ATC within each device is synchronized to within one cycle of one neighbor device's ATC.
- Synchronous operation enables the use of chip replication for the purpose of fault detection (see §2.6).
- An entire Vulcan network can be single-stepped by the mechanism outlined in §2.5.3.

2.4 Packet flow-control

The Vulcan method of packet transfer is related to *wormhole routing*. Here, we adopt Dally's distinctions between wormhole routing and virtual cut-through [12, 4]. In wormhole routing, each flit of a packet is advanced to the appropriate output port as soon is it arrives at a switching element input port. When the head of a packet is blocked, the flits are buffered in place. As soon as the output port is free, packet transfer resumes. The output port is chosen according to route information contained in one of the first flits (in Vulcan, the second flit) of the packet. In wormhole routing, flow-control is flit-based [4], not packet-based as in virtual cut-through [13].

Vulcan flow-control differs from wormhole routing in that packet flits are not typically buffered in place when a packet is blocked. In addition to small buffers at each input port, each switching element contains a relatively large shared buffer called the central queue.

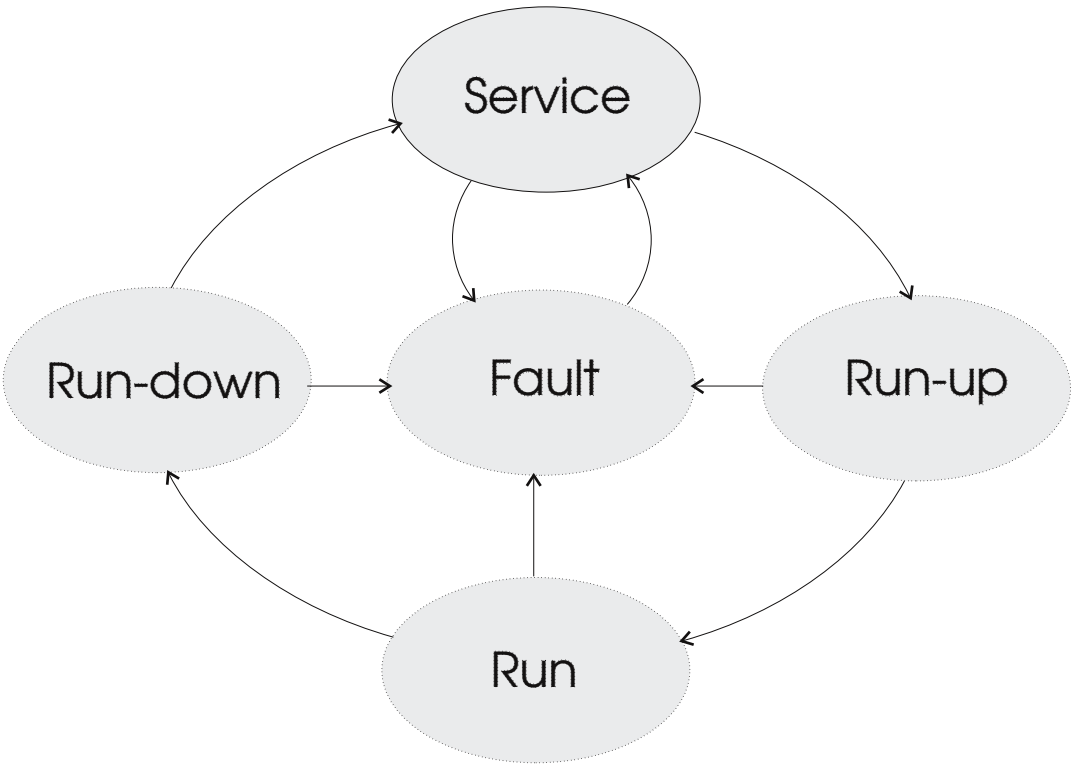

Figure 2: Vulcan mode transition diagram

The central queue is dynamically allocated to blocked packet flits from any input port. To highlight this departure from the standard wormhole definition, we call the Vulcan flow-control method *buffered wormhole routing*. Details of central queue allocation are covered in §3.4.

2.5 Unified data and service network

Most commercial multicomputers utilize one network for normal node-to-node (data) communication and another for network service. This adds to the complexity of the system design and reduces its reliability. In addition, data networks are optimized for high-performance communication. Due to cost considerations, service networks are typically not. Vulcan incorporates a unified data and service network, permitting network service and data operations to utilize the same high-speed network. The entire network operates in one of several *modes*, each mode encompassing a unique communication protocol. Figure 2 introduces the Vulcan modes. The major modes are Service mode for servicing the

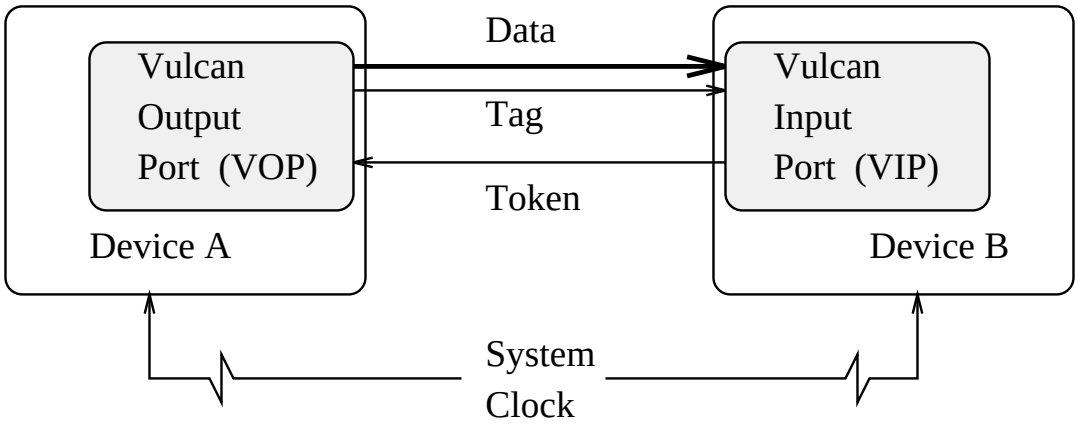

Figure 3: Vulcan I/O ports connected by a channel

Figure 4: A Vulcan run mode packet

network, and Run mode for normal data transfer. Service mode protocol will not be discussed here, except to note that this mode provides mechanisms for multicast and broadcast communication.

2.5.1 Run mode protocol

The Vulcan Run mode link protocol is token-based to accommodate link propagation delays of several cycles. As shown in Figure 3, each input port transmits tokens to the upstream output port. Each token represents the input port's readiness to accept one flit. An output port spends one token each time it transmits a flit. To maintain bandwidth for a link with a propagation delay of p cycles, the downstream input port must contain a buffer of at least size $2p$. The Vulcan prototype maintains bandwidth for $p \leq 15$.

An active Tag indicates a valid packet flit. An inactive Tag signals a control code. In the Vulcan prototype, two control codes are implemented: a null control code and a fault code. The use of the fault code is detailed in §2.6.

Packets vary in length up to 255 flits. The first flit holds the packet length, as shown in Figure 4. The next flit or flits contain route information. The rest of the packet is data. Each switching element examines the first route flit to determine the appropriate output

Figure 5: Service message format. Length = $n + 3$

port for packet forwarding. Used route flits are discarded (the length flit is correspondingly decremented). A packet contains no route flits when it reaches its destination.

Time is divided into 64-cycle frames. During the last 2 cycles of each frame, both Tag and Data carry CRC codes. The Token signal is encoded in two-cycle frames. A frame sequence of 0-1 represents no tokens, and 1-0 represents two tokens. The 0-0 and 1-1 encodings are invalid.

2.5.2 Service mode protocol

Service mode employs a simple circuit-switched protocol to reliably communicate with selected network devices. Devices are controlled by service nodes. Typically one service node controls the entire network, but large networks may utilize multiple service nodes to minimize service time.

Service nodes send service messages of the format shown in Figure 5. The command field identifies the type of service message. Each device on the current circuit executes the service command if the MID field matches the device's Message Identifier Register (MIR). The MIR is altered via the `load-MIR` message, and is reset to 0 in Fault mode. The execution of some service commands alters data flits (e.g., status query commands).

A subset of service messages provide circuit-switching operations. For example, the `set-pass-thru` message instructs a switching element to relay succeeding service messages to a specified set of output ports after it conditionally executes the message. Broadcast or multicast trees may be constructed. Switching elements provide two pass-thru circuits (forward and backward pass-thru) for the formation of loop circuits. Loop circuits pass information back to the service node.

2.5.3 Mode transitions

During normal operation, termed *run cycling*, devices cycle through the outer modes of Figure 2 in synchrony. The start time and duration of the next Run mode are chosen during Service mode and broadcast to all devices via service messages.

Run-down mode is a 16-cycle transitional mode that drains the links. Service mode always begins in an idle state. Run-up mode is a 16-cycle transitional mode that primes the links with the identical data present on the links at the end of the previous Run mode. To accomplish this, during Run and Run-up modes, each device continually buffers the

previous 16 cycles of transmitted data. This limits the propagation delay for a link to 15 cycles, because valid signals must have been sampled by every link input before Run mode begins.

Run cycling allows arbitrarily short Run modes—even 1 cycle— and thus provides the ability to single-step the entire network. Typically, the duration of Run mode is from a few seconds to several minutes. Service mode may be used to initiate service actions such as adding or removing a device, or to collect monitoring or debug information. Monitoring information can thus be collected without perturbing Run mode operation.

On network power up or after a network error is detected, the entire network is in Fault mode. Devices are transferred from Fault to Service mode when they receive a reserved sequence of tagged bytes at an input port.

2.6 Error detection

An objective in Vulcan is to detect and record the cause of every network error. Every switching element and most node elements of the Vulcan system are *shadowed*. For example, each switching element in the Vulcan network consists of two switch chips: one is designated the "master" and the other the "slave." Each output pin of a switch chip is implemented as a bidirectional I/O, and the identical output signals of master and slave are electrically connected. The master drives its signal outputs. The slave checks the value received on its corresponding inputs against the value it would have driven, were it the master. Miscompares are immediately reported.

The signals transmitted between networks elements cannot be checked by this shadowing methodology. Instead, all transmitted signals are checked with CRC codes. Relying upon shadowing and CRC codes (2.5.1, the attempt is thus made to catch and record every possible network error. This potentially frees node devices from verifying packet integrity.

An additional feature of Vulcan is *global faulting*. When an error is detected by any device, that device transfers to Fault mode (recall Figure 2) and immediately notifies its neighbors via a reserved control code termed the fault code. Each neighbor, in response, transfers to Fault mode and notifies its neighbors, until every device in the network is notified. Service software then collects the error status in each device and restarts normal network operation. The origination of the error can then be pinpointed by examining all error status information. Each network device records both the type and the time of error.

3 The Vulcan Switch Chip

The Vulcan Switch chip, illustrated in Figure 6, contains 8 receiver modules and 8 transmitter modules, an unbuffered crossbar, and the central queue. All ports are one flit (one byte) wide. In the absence of contention, packet flits incur 5 cycles of latency cutting through the chip via the crossbar path (§3.2). The chip is mounted in a 223 pin ceramic pin grid

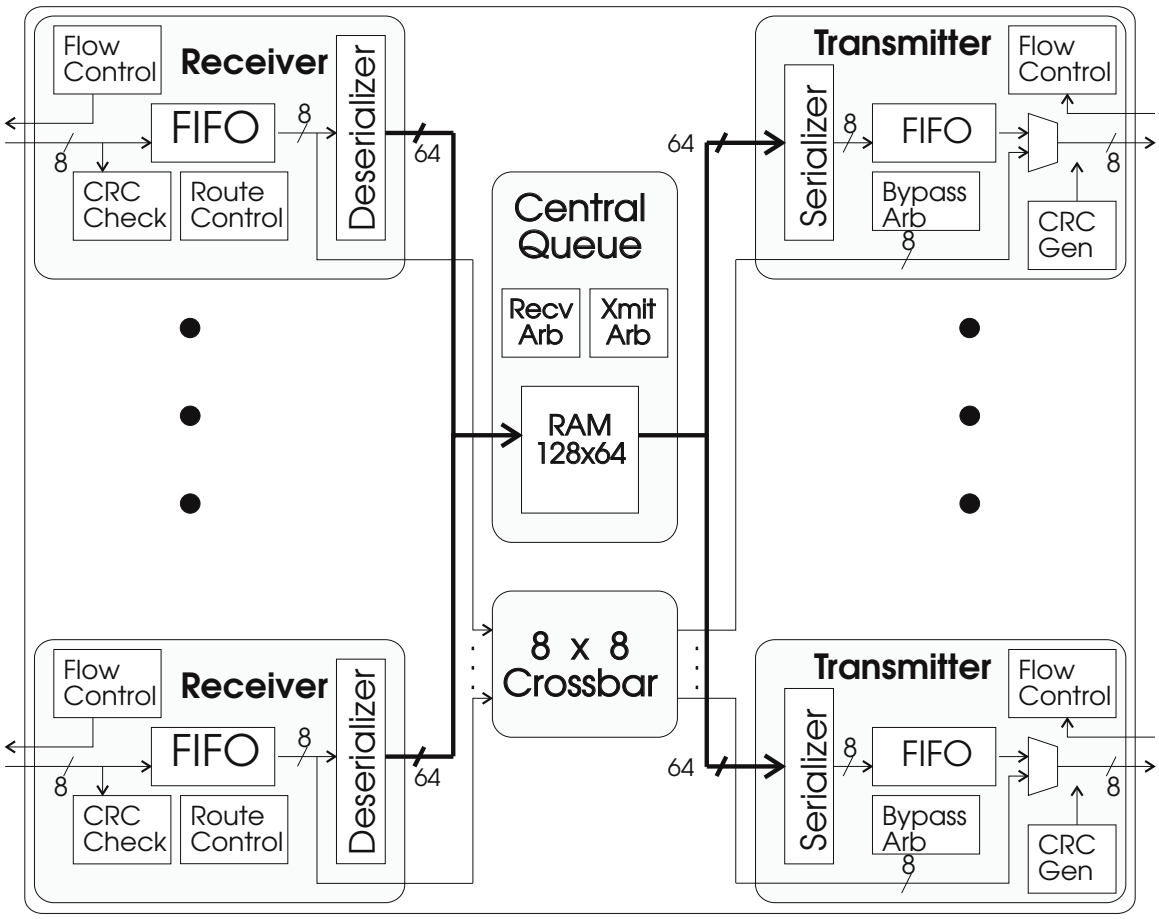

Figure 6: Vulcan Switch chip organization

array package, and is designed for 50 MHz operation. The design of this chip was originally targeted for the TF-1 machine [14].

3.1 Receivers

The switch chip contains eight identical receiver modules, one associated with each of the eight input ports. The receiver module performs five major functions: (1) administering the link flow-control protocol, (2) checking the link CRC codes, (3) buffering incoming data, (4) decoding packet routing information, and (5) deserializing incoming packets into 8-flit *chunks* when the packet is blocked. Buffering is accomplished with a 31-flit first-in-first-out (FIFO) queue. When an incoming packet encounters no contention for the selected output

port, packets are immediately forwarded via the Switch chip's crossbar. When the packet is blocked, 8-flit packet chunks are sent to the central queue for temporary buffering.

3.2 Crossbar Routing

The switch chip incorporates an unbuffered logical cross-bar that allows packets to pass directly from the receivers to the transmitters. These flit-serial crossbar paths permit packets to pass through the chip with low latency whenever there is no contention for the output port. As soon as a receiver decodes the routing information carried by an incoming packet, it asserts a crossbar request to the appropriate transmitter. If the crossbar request is not granted by the time the entire first chunk of the packet has been received, the crossbar request is dropped (and hence the packet will go to the central queue). Each transmitter arbitrates crossbar requests on a least-recently-served basis. A transmitter will honor no crossbar request if it is already transmitting a packet or if it has packet chunks stored in the central queue.

3.3 Transmitters

There are eight transmitter modules, one corresponding to each output port. When the central queue contains packet chunks destined for a transmitter, that transmitter requests the next packet chunk. Transmitter modules are served by the central queue in a least-recently-served fashion. As long as data is available, one transmitter is served each clock cycle. The transmitter accepts packet chunks from the central queue, serializes them, buffers them in a 7-flit output FIFO, and transmits them to the link in accordance with the link flow-control protocol. The transmitter is also responsible for computing and transmitting the CRC codes.

3.4 Central Queue

The central queue implements a buffered, time-multiplexed 8-way router. It accepts packet chunks from the receivers, stores them, and eventually passes them to the appropriate transmitters. The Vulcan central buffering scheme is similar to the centrally-buffered, dynamically-allocated switch described by Tamir and Frazier [15], but the Vulcan chip allocates central queue space on a chunk basis instead of a packet basis.

The central queue stores packets until they can be transmitted. The storage, a 128 by 64-bit dual-port RAM, holds up to 128 eight-flit packet chunks. The queue does not reserve a fixed amount of space for each output port; storage is allocated dynamically according to demand.

Stored packets are queued in FIFO order on eight linked lists, one list corresponding to each of the eight switch output ports. These lists are stored in a dual-port 128 by 7-bit RAM. Furthermore, for each packet, there is a linked list consisting of the constituent chunks. These lists are stored in another dual-port 128 by 7-bit RAM.

As long as queue space is available, one receiver can be served every cycle. As long as data is available, one transmitter is also served every cycle. Thus the bandwidth through the central queue matches the bandwidth into and out of the switch chip: eight flits per cycle.

To prevent starvation, receivers requesting service from the central queue are divided into two groups: those offering *critical* chunks and those offering *non-critical* chunks. A critical chunk is one that is ready for immediate forwarding to its destination transmitter. The first chunk of a new packet is critical if no other packets are queued for its transmitter. A continuation chunk is critical if all previous chunks of the packet have been forwarded to the transmitter. Chunks not immediately ready for forwarding are called non-critical.

One chunk of central queue storage is reserved for each transmitter. The remaining central queue storage is shared. Whenever a receiver has a non-critical chunk and space is available in the shared pool, the receiver requests service from the central queue. Whenever a receiver has a critical chunk, it requests service. Critical chunks are stored in the destination transmitter's reserved queue slot. Priority is assigned to the receivers on a least-recently-served basis.

If desired, the central queue storage can be partitioned into two pieces. In this *split* mode, half of the receiver and half of the transmitter ports are assigned to one piece; the other ports are assigned to the other piece. Split mode prevents congestion among one of the subsets of ports from interfering with traffic traversing the other subset.

4 The Vulcan Prototype

The Vulcan architecture is designed to scale to many thousands of nodes to support teraflop performance. The goal of the Vulcan prototype was to build the minimal set of hardware necessary to demonstrate all key components of Vulcan. To this target, the prototype comprised processor nodes, communication links, switch boards, I/O nodes, and a host node that provides a connection to a standard Micro Channel[1] slot. Full functionality of the system was achieved by early 1992. Figure 1 showed the components in a complete Vulcan system. The Vulcan system is modular and the configuration is subject to the intended job mix. For the targeted scientific applications the processor nodes would be the most replicated node. However, for applications such as database queries and transaction processing, one can envision significant numbers of I/O nodes. Each component of the system, including the two ASICs, was designed and simulated for 50 MHz operation. The Intel i860 was chosen as the processor because it was an off-the-shelf readily available single-chip microprocessor. The obtainable i860-XR was available only up to 40 MHz, and therefore the prototype is clocked at 40 MHz. As stated in §2, one Vulcan design criteria is to leverage the latest processor technology. The later time-frame of the SP1 allowed the use of the RISC System/6000 for the processor node. In this section we overview the major components of the system.

4.1 Link technology

All communication in Vulcan uses point-to-point bidirectional communication links that consist of two channels, each carrying data in opposite directions to their respective output port and input port pairs. For each channel, the actual transmission is via a set of 10 signal lines, 8 for data, one for Tag and one for the Token as was shown in Figure 3. For all cases where the link is driven off board, 100K ECL differential pairs are used. Such inter-board ECL signals have both source and sink terminators, custom designed to match the cable characteristics. Logic level translation for data bits is provided by a specially designed low-skew 20 bit buffer with 10 bits in each direction. The raw cable selected was an off-the-shelf 50 pin SCSI cable. The bidirectional link implementation required only 20 differential pairs to support the 2 sets of 10 signals, therefore providing 10 additional spare signals. The motivation was two-fold: (1) it gave us the ability to use the common SCSI cable components, and (2) the cable could support additional signals if the need arose. For instance, we have experimented with using these additional spares to distribute the clock via the data cables.

4.2 The MSMU Chip

The primary role of the MSMU (Memory Switch Management Unit) is to attach processors and I/O to the Vulcan switching network, and optionally serve as a DRAM controller. Communication services commonly found in the physical, datalink and network layers of most communication adapters are provided. One output port and one input port implement the link level protocol (i.e. tokens, tags, phase adjustment, etc.). Speed matching FIFO's consume and produce switch packets from/to the processor and I/O nodes. Three FIFO pairs are provided, two for Run mode communication and one for Service mode. Of the pairs provided for Run mode, one is 2K bytes in each direction, and the other is 512 bytes. The smaller pair provides an overtaking path within the MSMU, for both send and receive, on packet boundaries.

The MSMU provides a complete 8 byte pipelined i860 bus interface, and a 4 byte i386 bus interface (for the I/O node, §4.5). There is also a bus adapter which provides a mechanism for accessing i386 slave devices. There is a DRAM controller, and an inline ECC data path which provides four banks of fully interleaved DRAM (2MB or 8MB per bank). The memory, control registers, and speed matching FIFO's on the MSMU are all fully interleaved, and are accessed via a four level address pipe. The processor bus data rate 200 MB/S at 50 MHz. All DRAM refresh and initialization cycles are done by the MSMU.

The MSMU provides two other noteworthy functions. A "packet block move" facility allows the processor to initiate transfers directly between the user address space (MSMU DRAM) and the packet FIFOs. This reduces software copies of data via the processor, and permits a processor to overlap protocol execution with data movement. The other feature is called "service probe". This function gives remote access to any location in the MSMU's address space (e.g. system memory, control/status registers, FIFO's). This is useful for

diagnosing faults in a remote node whose processor is either unable to respond, or that may be untrustworthy. It is also used to IPL remote nodes as discussed in §5.3. The remaining MSMU functions are simple services such as timers, alarms, diagnostics, and status.

4.3 Processor Node

The processor node in Vulcan consists of a shadowed processor–MSMU pair and 32 megabytes of DRAM. The primary function of the processor node is to provide the horsepower required for the computationally intensive applications targeted to run on the system. In the design of the MSMU, if there were tradeoffs to be made in the support of host, I/O and processor nodes, priority was always given to processor node support. In a large Vulcan installation the processor node (commonly referred to as the compute node) would be the most replicated element. Therefore, in addition to performance, two prime design considerations were reliability and density.

In order to support reliability, the memory is protected with ECC. The error detection capability of the links was described in §2.6, and shadowing is used to detect errors on the actual computation. Shadowing was required and tricky for the compute node, because the microprocessor is an off-the-shelf component that was not designed to support such stringent synchronous operation. However, with careful design consideration of how we handle reset and the various control signals, we were able to successfully shadow the processor operation via the processor–MSMU pair.

To support the density requirement, a reliable packaging strategy that could support many processor nodes on a small footprint was necessary. A tub was designed to provide power, cooling, and communication for 16 processor nodes. In the prototype the processor logic card measures 8.5" x 14.5". The physical dimensions of this card could be reduced, but for ease of debug, we did not aim for the smallest card size possible.

4.4 Host Card

Attached to the Vulcan network is a service node that loads the processor nodes at boot time and provides diagnostic support during service mode. This service node is implemented by a RISC System/6000[1] with a plug-in Host card that connects to a Vulcan link. The card is a logical superset of the processor node, incorporating an MSMU, DRAM, and the microprocessor albeit without shadowing. In addition it allows the host system to control the node directly, as a surrogate for the on-card microprocessor. Thus the RISC System/6000 can communicate directly with the MSMU and DRAM.

The host card proved useful in hardware and software debug. For hardware debug, it provided a means to boot, monitor, and control a Vulcan processor node—the node embedded in the card—without going through a Vulcan link. Network protocols could be developed and tested in the AIX environment, rather than in the remote and spartan Vulcan processor node environment. For the operating system and applications, the host card provided a platform in which the microprocessor could be controlled directly and the

memory directly examined and modified. Host cards could be interconnected directly or via switching elements, with one card playing the host-card role and the others configured as nodes. These host-card "nodes" provide a direct window on node behavior for the prototyping of parallel code.

4.5 I/O Node

The I/O node employs the microprocessor, MSMU, and DRAM as found on the processor card. The I/O node contains additional hardware that allows the direct connection of SCSI peripheral devices. The node adds four SCSI I/O Processors to the on-card bus which is shared by the microprocessor and the MSMU. Special circuitry in the MSMU allows the NCR devices to communicate with the faster and wider 8-byte bus and the MSMU, and arbitration logic on the card time-slices the bus activity of the NCR devices, each of which is capable of independent bus master operation.

The Vulcan I/O node, as implemented in the prototype, consists of: a logic card, eight 400MB hard disk drives, and a printed circuit backplane into which the logic card and the disk drives are plugged. The logic card measures 18" x 7.5". It plugs into the backplane with two 88-pin connectors, through which all external signal and power connections are made. The disks drives are individually mounted on small carrier cards (7.5" x 4.5"), which plug into the backplane (20" x 5.5") with 64-pin connectors. The entire I/O node is mounted in a purpose-built metal frame.

4.6 Switch Board

The switch board implements a two stage network building block as illustrated in Figure 1. It provides any-to-any paths connecting 32 Vulcan links. All Switch chips are shadowed so the board contains 16 switch chips. The Switch chip is conceptually treated as an 8-ported device. The pin assignments are carefully chosen such that all the pins associated with ports 0-3 are on one half, ports 4-7 on the other. Taking advantage of this, ports 0-3 are used for inter-board links, and ports 4-7 for intra-board links. This minimizes the wire cross-overs on the board.

All inter-board signals, both data and clock, are driven as 100K ECL differential pairs. Additionally, on-board system and offset clock trees are maintained in ECL. Each switch chip requires a system clock and eight offset clocks for tuning. The printed circuit board measures 20" X 23", with 8 signal and 6 power layers.

4.7 Node-to-node performance

Using a minimal polling kernel on each processor node, we measured user to user message-passing latencies of approximately 2 microseconds for zero-byte messages, as illustrated in Figure 7. Figure 8 demonstrates the bandwidth measured on the prototype. Using the MSMU's "packet block move" capability we achieved an asymptotic bandwidth of 38.75

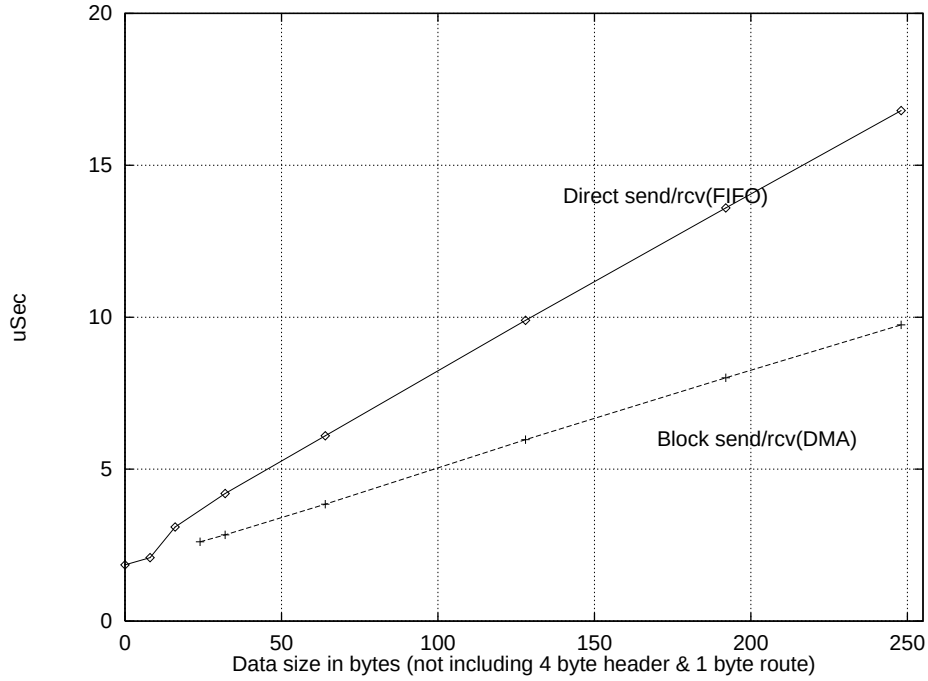

Figure 7: Latency measurements on the Vulcan prototype

Mbytes/sec, which matches the peak theoretical bandwidth of the links, considering CRC cycles. When the processor controls the packet transfer (direct send or receive), the attainable bandwidth is limited by the processor bus bandwidth required for software copying of packet data.

5 Support Software

5.1 Worm

The Worm package is the Vulcan network service software [16], and is also used on the SP1 system. It executes on a service node and controls the network during service mode. The Worm package provides initialization, tuning, global time synchronization, fault determination, and diagnostic services to the switching devices and all other network devices. It further supplies routines that facilitate direct control or interrogation of these devices.

The Worm is designed to service an arbitrary pre-specified network of Vulcan devices. The only inherent topology assumption is that each cable in the network is bidirectional. An input file describes the expected network topology. The Worm searches the network in a breadth-first manner, entering status information about each device and link in a

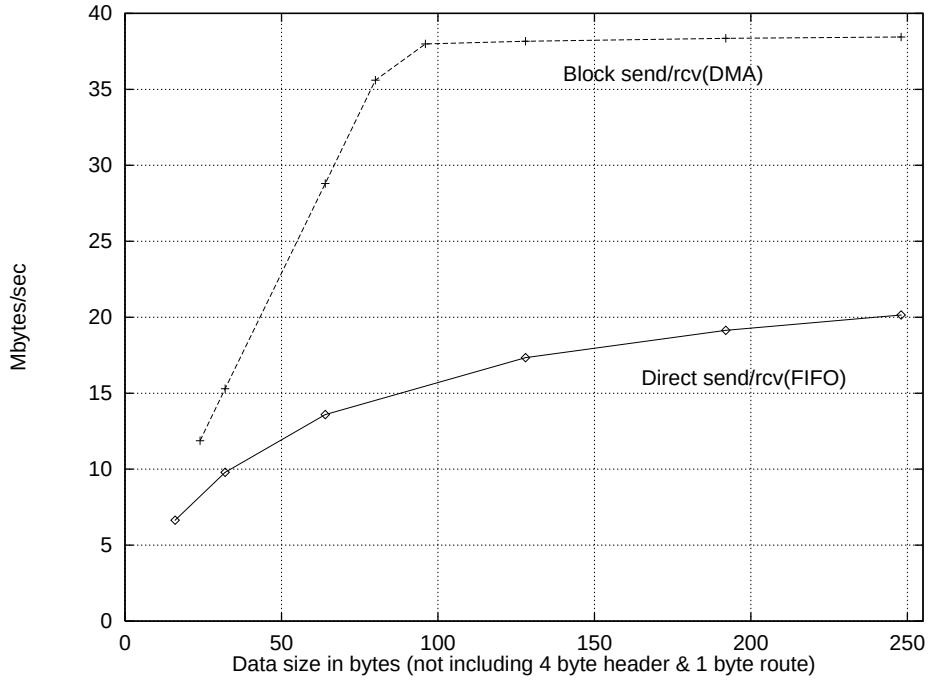

Figure 8: Bandwidth measurements on the Vulcan prototype

database. This database is subsequently used by the Vulcan Router (§5.2) to generate valid node-to-node routes.

A particularly crucial Worm function is tuning of link data arrival. Here we overview the tuning procedure.

Tuning

In the Vulcan prototype, all tuning is software-controlled, and utilizes delay logic at each Vulcan output port (VOP). In contrast, the Monarch parallel processor incorporated a hardwired method for tuning a serial link [2]. Two service commands enable the service software to synchronize data arrival for a channel. One command (tuning-control) adds an identical amount of delay to each of the VOP's Data and Tag outputs, and selects a separate delay adder for the VOP's Token input. A second command (cycle-counting) determines the number of cycles taken to: (a) send a pulse out of the VOP on a Data or Tag signal, (b) receive the pulse at the downstream input port, and *reflect* it back onto the Token signal, (c) receive the reflected pulse back on the VOP Token signal. "Reflection" at the input port is an action utilized specifically during tuning that: (1) clocks the 9 input signals into a register R, and (2) OR's the output of R into its Token output flip-flop.

A number of cycle-counting experiments—for carefully chosen delays—are performed to determine the best delay adder for the VOP's output and input signals for synchronizing signal reception. The simplified principle: as the Worm monotonically and incrementally increases the delay on an outgoing (or incoming) pulse, at some point the number of cycles recorded by a subsequent cycle-counting command will increase. At this point, the selected VOP tuning delays are very near the *worst* setting (near the set-up/hold region of receiving register logic.) Hence, the proper delay setting is $\approx \frac{1}{2}$ clock cycles away from the currently selected delay.

5.2 Route generation

System software obtains routes by table lookup. The table may contain multiple entries per source-destination pair to facilitate source-based random routing. The Vulcan Router produces this route table. Just as for the Worm, no topology is assumed. Instead, the Router uses the set of links and devices found acceptable by the Worm, and builds breadth-first spanning trees [17] using the resulting network graph to find shortest-path routes. A simple load-balancing strategy is used to ensure that links are included in node-to-node paths in a balanced manner.

5.3 Node IPL

We use the broadcast capability of service mode to quickly initialize every processor and I/O node and to load the operating system. The microprocessor is suspended via a `suspend` service command or by Fault mode—either action causes the MSMU to issue a hold request to the processor. The processor is not resumed until the MSMU receives a `resume` service command. Meanwhile, the service node broadcasts `probe-write` commands containing the operating system image. These commands instruct the MSMU to load DRAM with the received data.

6 Lessons learned

The Vulcan prototype successfully demonstrates a number of Vulcan's unusual design properties. Tuning of data works flawlessly and frees system integrators from careful planning of cable lengths for clock distribution or data transfer. The unified communication network simplifies system integration, increases reliability, and speeds network service operations. Global time, synchronous operation, shadowing, and global faulting have proved invaluable in fault determination.

In comparison, other features have not been as useful. For example, the powerful ability to single-step the entire network was little used in hardware bringup. It is perhaps overkill considering today's sophisticated chip and system simulation tools. Our strategy for interleaving Run and Service mode provides the ability to monitor Run mode without perturbing

Run mode, but we must preplan each entry into Service mode. It could be useful to send service messages at any time, even if it perturbs Run mode operation.

The most contentious architectural questions involve the global faulting mechanism. Given a sufficiently large and unreliable network, it is possible to spend a significant percentage of time recovering from network errors. Hence, it can be argued that global faulting is not scalable, and that it should be replaced with local fault handling mechanisms. Conversely, global faulting has proved advantageous in simplifying the design of the fault location and isolation features of the "Worm" service software. It provides a record of every error and its type.

As cycle times decrease, it will become more difficult to apply our method of data tuning, which assigns a single delay adjustment to all transmitted bits of data. For our 50 MHz parts, a single delay setting is not troublesome even for cables over 100 feet in length. Systems running at much higher bandwidths, however, would likely be forced to use exotic interconnect technology or tune bits on an individual basis.

The transition

In 1992, the Highly Parallel Supercomputing Systems Laboratory (HPSSL) was established in Kingston, N.Y. to develop a scalable parallel computing family of systems. The working Vulcan prototype was an influential force behind IBM's decision to enter this market. The initial HPSSL offering is the SP1.

The communication subsystem of the SP1—the High Performance Switch—is essentially Vulcan prototype technology. It utilizes the link and switching technology described in §3, §4.1, §4.2, and §4.6, including the MSMU and Switch chips. The SP1 network service strategy is based upon the Vulcan service and support software overviewed in §5.1 and §5.2. Each SP1 processing node is a RISC System/6000 processor. The initial 1993 HPSSL SP1 product offering provided system sizes from 16 to 64 nodes, although configurations of 128-way are installed and operational at national labs and universities, and larger systems up to 512-way will be installed in 1994.

Acknowledgements

The authors wish to thank colleagues at the IBM Thomas J. Watson Research Center. Without their countless contributions, the Vulcan prototype would never have become reality. In particular, senior management played a significant role in the direction and support of this effort. Zeev Barzilai formed the Parallel Systems department, provided strategic direction and invaluable support of Vulcan, and successfully transferred this technology to IBM's first line of scalable parallel computers. Marc Snir provided leadership and contributions in parallel software. Winfried Wilcke (formerly of IBM), Randy Moulic, and C.J. Tan provided leadership and contributions to the hardware. Bob Lemay and Rich Swetz contributed in several ways to the packaging, clocking, and technology of Vulcan.

References

[1] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, "The network architecture of the Connection Machine CM-5," in *Proc. 1992 Symp. Parallel Algorithms and Architectures*, pp. 272–285, ACM, 1992.

[2] R. D. Rettberg, W. R. Crowther, P. P. Carvey, and R. S. Tomlinson, "The Monarch parallel processor hardware design," *IEEE Computer*, vol. 23, pp. 18–30, April 1990.

[3] M. Noakes and W. J. Dally, "System design of the J-machine," in *Proc. Sixth MIT Conf. Advanced Research in VLSI*, pp. 179–194, MIT Press, 1990.

[4] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[5] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. on Computers*, vol. C-24, pp. 1145–1155, Dec. 1975.

[6] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Ann. Symp. on Computer Architecture*, pp. 21–28, 1973.

[7] M. C. Pease, III, "The indirect binary n-cube microprocessor array," *IEEE Trans. on Computers*, vol. C-26, pp. 458–473, May 1977.

[8] G. J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons.* New York, NY: Wiley & Sons, 1987.

[9] I. D. Scherson and C.-H. Chien, "Least common ancestor networks," in *7th Int. Parallel Processing Symp.*, pp. 507–513, 1993.

[10] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. on Computers*, vol. C-34, pp. 892–901, Oct. 1985.

[11] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558–565, July 1978.

[12] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Computers*, vol. C-36, pp. 547–553, May 1987.

[13] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communications switching technique," *Computer Networks*, vol. 3, pp. 267–286, Sept. 1979.

[14] M. M. Denneau, P. H. Hochschild, and G. Schichman, "The switching network of the TF-1 parallel supercomputer," *Supercomputing Magazine*, pp. 7–10, Winter 1988.

[15] Y. Tamir and G. L. Frazier, "Hardware support for high-priority traffic in VLSI communication switches," *J. of Parallel and Distributed Computing*, vol. 14, pp. 402–416, April 1992.

[16] C. B. Stunkel, "The Vulcan Worm," tech. rep., IBM T. J. Watson Research Center, 1994.

[17] E. M. Reingold, J. Nievergelt, and N. Deo, *Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice-Hall, 1977.