Architecture and Implementation of MEMORY CHANNEL 2

Marco Fillo
Richard B. Gillett

The MEMORY CHANNEL network is a dedicated cluster interconnect that provides virtual shared memory among nodes by means of internodal address space mapping. The interconnect implements direct user-level messaging and guarantees strict message ordering under all conditions, including transmission errors. These characteristics allow industry-standard communication interfaces and parallel programming paradigms to achieve much higher efficiency than on conventional networks. This paper presents an overview of the MEMORY CHANNEL network architecture and describes DIGITAL's crossbar-based implementation of the second-generation MEMORY CHANNEL network, MEMORY CHANNEL 2. This network provides bisection bandwidths of 1,000 to 2,000 megabytes per second and a sustained process-to-process bandwidth of 88 megabytes per second. One-way, process-to-process message latency is less than 2.2 microseconds.

In computing, a cluster is loosely defined as a parallel system comprising a collection of stand-alone computers (each called a node) connected by a network. Each node runs its own copy of the operating system, and cluster software coordinating the entire parallel system attempts to provide users with a unified system view. Since each node in the cluster is an off-the-shelf computer system, clusters offer several advantages over traditional massively parallel processors (MPPs) and large-scale symmetric multiprocessors (SMPs). Specifically, clusters provide[1]

- Much better price/performance ratios, opening a wide range of computing possibilities for users who could not otherwise afford a single large system.
- Much better availability. With appropriate software support, clusters can survive node failures, whereas SMP and MPP systems generally do not.
- Impressive scaling (hundreds of processors), when the individual nodes are medium-scale SMP systems.
- Easy and economical upgrading and technology migration. Users can simply attach the latest-generation node to the existing cluster network.

Despite their advantages and their impressive peak computational power, clusters have been unable to displace traditional parallel systems in the marketplace because their effective performance on many real-world parallel applications has often been disappointing. Clusters' lack of computational efficiency can be attributed to their traditionally poor communication, which is a result of the use of standard networking technology as a cluster interconnect. The development of the MEMORY CHANNEL network as a cluster interconnect was motivated by the realization that the gap in effective performance between clusters and SMPs can be bridged by designing a communication network to deliver low latency and high bandwidth all the way to the user applications.

Over the years, many researchers have recognized that the performance of the majority of real-world parallel applications is affected by the latency and bandwidth available for communication.[2–5] In particular, it has been shown[2,6,7] that the efficiency of parallel scientific applications is strongly influenced by the

system's architectural balance as quantified by its communication-to-computation ratio, which is sometimes called the q-ratio.[2] The q-ratio is defined as the ratio between the time it takes to send an 8-byte floating-point result from one process to another (communication) and the time it takes to perform a floating-point operation (computation). In a system with a q-ratio equal to 1, it takes the same time for a node to compute a result as it does for the node to communicate the result to another node in the system. Thus, the higher the q-ratio, the more difficult it is to program a parallel system to achieve a given level of performance. Q-ratios close to unity have been obtained only in experimental machines, such as iWarp[8] and the M-Machine,[9] by employing direct register-based communication.

Table 1 shows actual q-ratios for several commercial systems.[10,11] These q-ratios vary from about 100 for a DIGITAL AlphaServer 4100 SMP system using shared memory to 30,000 for a cluster of these SMP systems interconnected over a fiber distributed data interface (FDDI) network using the transmission control protocol/internet protocol (TCP/IP). An MPP system, such as the IBM SP2, using the Message Passing Interface (MPI) has a q-ratio of 5,714. The MEMORY CHANNEL network developed by Digital Equipment Corporation reduces the q-ratio of an AlphaServer-based cluster by a factor of 38 to 82 to be within the range of 367 to 1,067. Q-ratios in this range permit clusters to efficiently tackle a large class of parallel technical and commercial problems.

The benefits of low-latency, high-bandwidth networks are well understood.[12,13] As shown by many studies,[14,15] high communication latency over traditional networks is the result of the operating system overhead involved in transmitting and receiving messages. The MEMORY CHANNEL network eliminates this latency by supporting direct process-to-process communication that bypasses the operating system. The MEMORY CHANNEL network supports this type of communication by implementing a natural extension of the virtual memory space, which provides direct, but protected, access to the memory residing in other nodes.

Based on this approach, DIGITAL developed its first-generation MEMORY CHANNEL network (MEMORY CHANNEL 1),[16] which has been shipping in production since April 1996. The network does not require any functionality beyond the peripheral component interconnect (PCI) bus and therefore can be used on any system with a PCI I/O slot. DIGITAL currently supports production MEMORY CHANNEL clusters as large as 8 nodes by 12 processors per node (a total of 96 processors). One of these clusters was presented at Supercomputing '95 and ran clusterwide applications using High Performance Fortran (HPF),[4] Parallel Virtual Machine (PVM),[17] and MPI[18] in DIGITAL's Parallel Software Environment (PSE). This 96-processor system has a q-ratio of 500 to 1,000, depending on the communication interface. A 4-node MEMORY CHANNEL cluster running DIGITAL TruCluster software[19] and the Oracle Parallel Server has held the cluster performance world record on the TPC-C benchmark[20]—the industry standard in on-line transaction processing—since April 1996.

We next present an overview of the generic MEMORY CHANNEL network to justify the design goals of the second-generation MEMORY CHANNEL network (MEMORY CHANNEL 2). Following this overview, we describe in detail the architecture of the two components that make up the MEMORY CHANNEL 2 network: the hub and the adapter. Last, we present hardware-measured performance data.

MEMORY CHANNEL Overview

The MEMORY CHANNEL network is a dedicated cluster interconnection network, based on Encore's

Table 1
Comparison of Communication and Computation Performance (q-ratio) for Various Parallel Systems

System	Communication Performance Latency (Microseconds)	Computation Performance Based on LINPACK 100 × 100 (Microseconds/FLOP)	Communication-to-computation Ratio (q-ratio)
AlphaServer 4100 Model 300 configurations			
SMP using shared memory messaging	0.6	0.006	100
SMP using MPI	3.4	0.006	567
FDDI cluster using TCP/IP	180.0	0.006	30,000
MEMORY CHANNEL cluster using native messaging	2.2	0.006	367
MEMORY CHANNEL cluster using MPI	6.4	0.006	1,067
IBM SP2 using MPI	40.0	0.006	5,714

MEMORY CHANNEL technology, that supports virtual shared memory space by means of internodal memory address space mapping, similar to that used in the SHRIMP system.[21] The MEMORY CHANNEL substrate is a flat, fully interconnected network that provides *push*-only message-based communication.[16,22] Unlike traditional networks, the MEMORY CHANNEL network provides low-latency communication by supporting direct user access to the network. As in Scalable Coherent Interface (SCI)[23] and Myrinet[24] networks, connections between nodes are established by mapping part of the nodes' virtual address space to the MEMORY CHANNEL interface.

A MEMORY CHANNEL connection can be opened as either an outgoing connection (in which case an address–to–destination node mapping must be provided) or an incoming connection. Before a pair of nodes can communicate by means of the MEMORY CHANNEL network, they must consent to share part of their address space—one side as outgoing and the other as incoming. The MEMORY CHANNEL network has no storage of its own. The granularity of the mapping is the same as the operating system page size.

MEMORY CHANNEL Address Space Mapping

Mapping is accomplished through manipulation of page tables. Each node that maps a page as incoming allocates a single page of physical memory and makes it available to be shared by the cluster. The page is always resident and is shared by all processes in the node that map the page. The first map of the page causes the memory allocation, and subsequent reads/maps point to the same page. No memory is allocated for pages mapped as outgoing. The mapper simply assigns the page table entry to a portion of the MEMORY CHANNEL hardware transmit window and defines the destination node for that transmit subspace. Thus, the amount of physical memory consumed for the clusterwide network is the product of the operating system page size and the total number of pages mapped as incoming on each node.

After mapping, MEMORY CHANNEL accesses are accomplished by simple load and store instructions, as for any other portion of virtual memory, without any operating system or run-time library calls. A store instruction to a MEMORY CHANNEL outgoing address results in data being transferred across the MEMORY CHANNEL network to the memory allocated on the destination node. A load instruction from a MEMORY CHANNEL incoming channel address space results in a read from the local physical memory initialized as a MEMORY CHANNEL incoming channel. The overhead (in CPU cycles) in establishing a MEMORY CHANNEL connection is much higher than that of using the connection. Because of the memory-mapped nature of the interface, the transmit or receive overhead is similar to an access to local main memory. This mechanism is the fundamental reason for the low MEMORY CHANNEL latency. Figure 1 illustrates an example of MEMORY CHANNEL address mapping.

The figure shows two sets of independent connections. Node 1 has established an outgoing channel to node 3 and node 4 and also an incoming channel to itself. Node 4 has an outgoing channel to node 2.

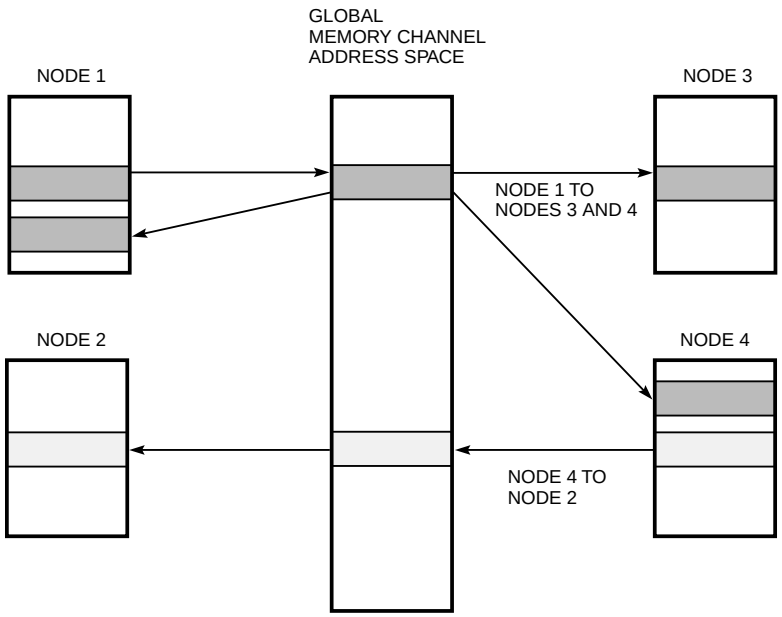


Figure 1
MEMORY CHANNEL Mapping of a Portion of the Clusterwide Address Space

All connections are unidirectional, either outgoing or incoming. To map a channel as both outgoing and incoming to the same shared address space, node 1 maps the channel two times into a single process' virtual address space. The mapping example in Figure 1 requires a total of four pages of physical memory, one for each of the four arrows pointed toward the nodes' virtual address spaces.

MEMORY CHANNEL mappings reside in two page control tables (PCTs) located on the MEMORY CHANNEL interface, one on the sender side and one on the receiver side. As shown in Figure 2, each page entry in the PCT has a set of attributes that specify the MEMORY CHANNEL behavior for that page.

The page attributes on the sender side are

- Transmit enabled, which must be set to allow transmission from store instructions to a specific page
- Local copy on transmit, which directs an ordered copy of the transmitted packet to the local memory
- Acknowledge request, which is used to request acknowledgments from the receiver node
- Transmit enabled under error, which is used in error recovery communication
- Broadcast or point-to-point, which defines the type of packet to all nodes or to a single node in the cluster
- Request acknowledge, which requests a reception acknowledgment from the receiver

The page attributes on the receiver side are

- Receive enabled, which must be set to allow reception of messages addressed to a specific virtual page
- Interrupt on receive, which generates an interrupt on reception of a packet
- Receive enabled under error, which is asserted for error recovery communication pages
- Remote read, which identifies all packets that arrive at a page as requests for a remote read operation
- Conditional write, which identifies all packets that arrive at a page as conditional write packets

MEMORY CHANNEL Ordering Rules

The MEMORY CHANNEL communication paradigm is based on three fundamental ordering rules:

1. Single-sender Rule: All destination nodes will receive packets in the order in which they were generated by the sender.
2. Multisender Rule: Packets from multiple sender nodes will be received in the same order at all destination nodes.
3. Ordering-under-errors Rule: Rules 1 and 2 must apply even when an error occurs in the network.

Let $Pj_{M\to X}$ be the jth point-to-point packet from a sender node M to a destination node X, and let Bj_{M} be the jth broadcast packet from node M to all other nodes. If node M sends the following sequence of packets,

$$\underset{\text{(last)}}{P2_{M\to X}}, P1_{M\to Y}, B1_{M}, \underset{\text{(first)}}{P1_{M\to X}},$$

Rule 1 dictates that nodes X and Y will receive the packets in the following order:

at node X, $\underset{\text{(last)}}{P2_{M\to X}}, B1_{M}, \underset{\text{(first)}}{P1_{M\to X}}$

at node Y, $\underset{\text{(last)}}{P1_{M\to Y}}, \underset{\text{(first)}}{B1_{M}}.$

If a node N is also sending a sequence of packets, in the following order,

$$\underset{\text{(last)}}{P3_{N\to X}}, P2_{N\to X}, B2_{N}, P2_{N\to Y}, B1_{N}, P1_{N\to Y}, \underset{\text{(first)}}{P1_{N\to X}},$$

there is a finite set of valid reception orders at destination nodes X and Y, depending on the actual arrival time of the requests to the point of global ordering. Rule 1 dictates that all packets from node M (or N) to node X (or Y) must arrive at node X (or Y) in the order in which they were transmitted. Rule 2 dictates that, regardless of the relative order among the senders, messages destined to both receivers must be received in the same order. For example, if X receives $B2_{N}$, $B1_{M}$, and $B1_{N}$, then Y should receive these packets in the

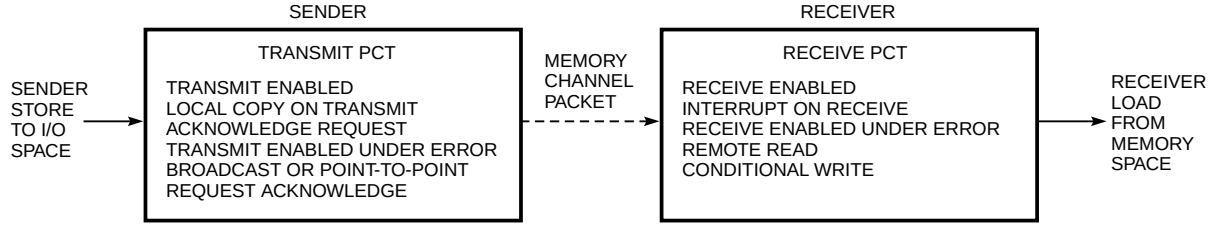


Figure 2
MEMORY CHANNEL Page Control Attributes

same order. One arrival order congruent with both of these rules is the following:

at node X,
$P3_{N\to X}$, $P2_{N\to X}$, $P2_{M\to X}$, $B2_{N}$, $B1_{M}$, $B1_{N}$, $P1_{N\to X}$, $P1_{M\to X}$
(last) (first)

at node Y,
$B2_{N}$, $P2_{N\to Y}$, $P1_{M\to Y}$, $B1_{M}$, $B1_{N}$, $P1_{N\to Y}$.

These rules are independent of a particular interconnection topology or implementation and must be obeyed in all generations of the MEMORY CHANNEL network.

On the MEMORY CHANNEL network, error handling is a shared responsibility of the hardware and the cluster management software. The hardware provides real-time precise error handling and strict packet ordering by discarding all packets in a particular path that follow an erroneous one. The software is responsible for recovering the network from the faulty state back to its normal state and for retransmitting the lost packets.

Additional MEMORY CHANNEL Network Features

Three additional features of the MEMORY CHANNEL network make it ideal for cluster interconnection:

1. A hardware-based barrier acknowledge that sweeps the network and all its buffers
2. A fast, hardware-supported lock primitive
3. Node failure detection and isolation

Because of the three ordering rules, the MEMORY CHANNEL network acknowledge packets are implemented with little variation over ordinary packets. To request acknowledgment of packet reception, a node sends an ordinary packet marked with the request-acknowledge attribute. The packet is used to sweep clean the network queues in the sender destination path and to ensure that all previously transmitted packets have reached the destination. In response to the reception of a MEMORY CHANNEL acknowledge request, the destination node transmits a MEMORY CHANNEL acknowledgment back to the originator. The arrival of the acknowledgment at the originating node signals that all preceding packets on that path have been successfully received.

MEMORY CHANNEL locks are implemented using a lock-acquire software data structure mapped as both incoming and outgoing by all nodes in the cluster. That is, each node will have a local copy of the page kept coherent by the mapping. To acquire a lock, a node writes to the shared data structure at an offset corresponding to its node identifier. MEMORY CHANNEL ordering rules guarantee that the write order to the data structure—including the update of the copy local to the node that is setting the lock—is the same for all nodes. The node can then determine if it was the only bidder for the lock, in which case the node has won the lock. If the node sees multiple bidders for the same lock, it resorts to an operating system–specific back-off-and-retry algorithm. Thanks to the MEMORY CHANNEL guaranteed packet ordering, even under error the above mechanism ensures that at most one node in the cluster perceives that it was the first to write the lock data structure. To guarantee that data structures are never locked indefinitely by a node that is removed from a cluster, the cluster manager software also monitors lock acquisition and release.

The MEMORY CHANNEL network supports a strong-consistency shared-memory model due to its strict packet ordering. In addition, the I/O operations used to access the MEMORY CHANNEL are fully integrated within the node's cache coherency scheme. Besides greatly simplifying the programming model, such consistency allows for an implementation of spinlocks that does not saturate the memory system. For instance, while a receiver is polling for a flag that signals the arrival of data from the MEMORY CHANNEL network, the node processor accesses only the locally cached copy of the flag, which will be updated whenever the corresponding main memory location is written by a MEMORY CHANNEL packet.

Unlike other networks, the MEMORY CHANNEL hardware maintains information on which nodes are currently part of the cluster. Through a collection of timeouts, the MEMORY CHANNEL hardware continuously monitors all nodes in the cluster for illegal behavior. When a failure is detected, the node is isolated from the cluster and recovery software is invoked. A MEMORY CHANNEL cluster is equipped with software capable of reconfiguration when a node is added or removed from the cluster. The node is simply brought on-line or off-line, the event is broadcast to all other nodes, and operation continues. To provide tolerance to network failures, the cluster can be equipped with a pair of topologically identical MEMORY CHANNEL networks, one for normal operation and the other for failover. That is, when a nonrecoverable error is detected on the active MEMORY CHANNEL network, the software switches over to the standby network, in a manner transparent to the application.[19]

The First-generation MEMORY CHANNEL Network

The first generation of the MEMORY CHANNEL network consists of a node interface card and a concentrator or hub. The interface card, called an adapter, plugs into the I/O PCI. To send a packet, the CPU

writes to the portion of I/O space mapped to the PCI bus. The store-to-memory is handled by the node's PCI interface device, which initiates a PCI transfer targeting the MEMORY CHANNEL adapter transmit window. When a message is received, the MEMORY CHANNEL adapter initiates a PCI transfer to write to the node's CPU memory, targeting the node's PCI interface, which then accesses the node's main memory.

Besides writing to the node's CPU, an I/O device on the PCI bus can transmit directly to a MEMORY CHANNEL adapter. This allows, for example, a disk controller to transfer data directly from the disk to a remote node's memory. The data transfer does not affect the host system's memory bus. The design choice of interfacing MEMORY CHANNEL to the PCI bus instead of directly to the node CPU bus is not an architectural one, but rather one of practicality and universality. The PCI is available on most of today's systems of varying performance and size and is, therefore, an ideal interface point that allows hybrid clusters to be built. The obvious disadvantages of a peripheral interface bus are the additional latency incurred because of the extra CPU-to-PCI hop and a possible limitation on the available bus bandwidth.

The MEMORY CHANNEL 1 hub is a broadcast-only shared bus capable of interconnecting up to eight nodes. The MEMORY Channel 1 adapters and the hub are interconnected in a star topology via 37-bit-wide (32 bits of data plus sideband signals) half-duplex channels. The cables can be up to 4 meters long, and the signaling level is 5-volt TTL. A two-node cluster can be formed without employing a hub, by direct node-to-node interconnection. This configuration is also known as virtual hub configuration.

The current release of the MEMORY CHANNEL 1 hardware achieves a sustained point-to-point bandwidth of 66 megabytes per second (MB/s) (from user process to user process). Maximum sustained broadcast bandwidth is also 66 MB/s (from a user process to many user processes). The cross-section MEMORY CHANNEL 1 hub bandwidth is 77 MB/s. Small message latency is 2.9 microseconds (μs) (from a sender process STORE instruction to a message LOAD by a receiver process). The processor overhead is less than 150 nanoseconds (ns) for a 32-byte packet (which is also the largest packet size).

As demonstrated in the literature, standard message-passing application programming interfaces (APIs) benefit greatly from these MEMORY CHANNEL communication capabilities.[12,17,25] MPI, PVM, and HPF on MEMORY CHANNEL 1 all have one-way message latencies of less than 10 μs. These latency numbers are more than a factor of five lower than those for traditional MPP architectures (52 to 190 μs).[11] Communication performance improvements of this magnitude translate into cluster performance gains of 25 to 500 percent.[12]

MEMORY CHANNEL 2 Architecture

Based on the experience with the first-generation product, the design goals for MEMORY CHANNEL 2 were twofold: (1) yield a significant performance improvement over MEMORY CHANNEL 1, and (2) provide functional enhancements to extend hardware support to new operating systems and programming paradigms.

The MEMORY CHANNEL 2 performance/hardware enhancement goals were

- Network bisection bandwidth scalable with the number of nodes: 1,000 MB/s for an 8-node cluster and 2,000 MB/s for a 16-node cluster
- Improved point-to-point bandwidth, exploiting the maximum capability of the 32-bit PCI bus: 97 MB/s for 32-byte packets and 127 MB/s for 256-byte packets
- Full-duplex channels to allow simultaneous bidirectional transfers
- Maximum copper cable length of 10 meters (increased from 4 meters on MEMORY CHANNEL 1) and fiber support up to 3 kilometers
- A link layer communication protocol compatible with future generations of MEMORY CHANNEL hardware and optical fiber interconnections
- Enhanced degree of error detection

The MEMORY CHANNEL 2 functional/software enhancement goals were

- Software compatible with the first-generation MEMORY CHANNEL hardware
- Receive-side address remapping and variable page size to better support new operating systems, such as Windows NT, and non-Alpha microprocessors
- Remote read capabilities
- Global time synchronization mechanism
- Conditional write access to support a faster recoverable messaging

These two sets of requirements translate into architectural and technological constraints that define the MEMORY CHANNEL 2 design space. To increase the bisection bandwidth, the hub had to implement an architecture that supported concurrent transfers. On MEMORY CHANNEL 1, all senders must arbitrate for the same hub resource (the bus) on every data transfer. Every data transmission occupies the entire MEMORY CHANNEL 1 hub for the duration of its

transfer, and all message filtering is performed by the receivers. Substantial network traffic causes congestion because all sender nodes fight for the same resource. This congestion results in a decrease in the communication speed and thus an increase in the effective q-ratio as seen by the applications.

On MEMORY CHANNEL 2, the hub has been designed as an *N*-by-*N* nonblocking full-duplex crossbar with broadcast capabilities, with $N = 8$ or $N = 16$. Such an architecture provides a bisection bandwidth that scales with the number of nodes and thus remains matched to the point-to-point bandwidth of the individual channels while avoiding congestion among independent communication paths. Therefore, an increase in network traffic will have little effect on the effective q-ratio.

The MEMORY CHANNEL ordering rules are easily met on a crossbar of this type, as follows:

1. The single-sender ordering rule is naturally obeyed by the fact that the architecture provides a single path from any source to any destination.
2. The multisender ordering rule is enforced by taking over all the crossbar routing resources during broadcast. Although less efficient than broadcast by packet replication, this technique ensures a strict common ordering for all destinations.

Finally, crossbar switches are practical to implement for a modest number of nodes (8 to 32), but given the availability of medium-size SMPs, they provide a satisfactory degree of scaling for the great majority of practical clustering applications. For instance, cluster technology can easily provide a 1,000-processor system simply by connecting 32 nodes, each one a 32-way SMP.

The requirement for a higher point-to-point bandwidth called for a shift from half-duplex to full-duplex links. A longer cable length imposed the choice of a signaling technique other than the TTL employed in the MEMORY CHANNEL 1 network. The design team adopted low-voltage differential signaling (LVDS)[26] as the signaling technique for the second and future generations of the MEMORY CHANNEL network on copper. One of the major decisions that faced the team was whether to maintain the parallel channel of MEMORY CHANNEL 1 or to adopt a serial channel to minimize skew transmission problems for large communication distances. The bandwidth demands of future cluster nodes indicated that serial links would not provide sufficient bandwidth expansion capabilities at reasonable cost. Thus, the channel data path width was chosen to be 16 bits, a suitable compromise that would offer a manageable channel-to-channel skew while providing the required bandwidth. Figure 3 illustrates the distinctions between the first- and second-generation MEMORY CHANNEL architectures.

MEMORY CHANNEL 2 Link Protocol

The MEMORY CHANNEL 2 communication protocol was engineered with the goal of ensuring compatibility with optical fiber's unidirectional medium. The interconnection substrate consists of a pair of unidirectional channels, one incoming and one outgoing. Each channel consists of a 16-bit data path, a framing signal, and a clock. The channel carries two types of packets: data and control. Data packets vary in size and carry application data. Control packets are used to exchange flow control, port state, and global clock information. Control packets take priority over data packets. They are inserted immediately when flow control state change is needed and, otherwise, are generated on a regular interval (millisecond) to update less time-critical state. The MEMORY CHANNEL 2 data packet format is shown in Figure 4a. The header of the data packet contains a packet type (TP), a destination identifier (DNID), a remote command (CMD), and a sender identifier (SID). The data payload starts with the destination address and can vary in length from 4 to 256 bytes (two to one hundred twenty-eight 16-bit cycles). It is followed by two 16-bit cycles of Reed-Solomon error detection code.

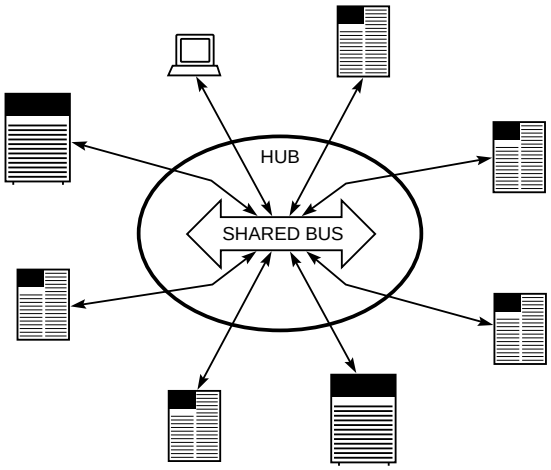
The control packet format is shown in Figure 4b. The packet is identified by a distinct TP and carries network and flow control information such as port status (PSTAT), configuration (CFG), DNID, hub status, and global status.

Similar to MEMORY CHANNEL 1, MEMORY CHANNEL 2 uses a clock-forwarding technique in which the transmit clock is sent along with the data and is used at the receiver to recover the data. Data is transmitted on both edges of the forwarded clock, and a novel dynamic retiming technique is used to synchronize the incoming packets to the node's local clock. The retiming circuit locks onto a good sample of the incoming data at the start of every packet and ensures accurate synchronization for the packet duration, as long as predefined conditions on maximum packet size and clock drifts are maintained.

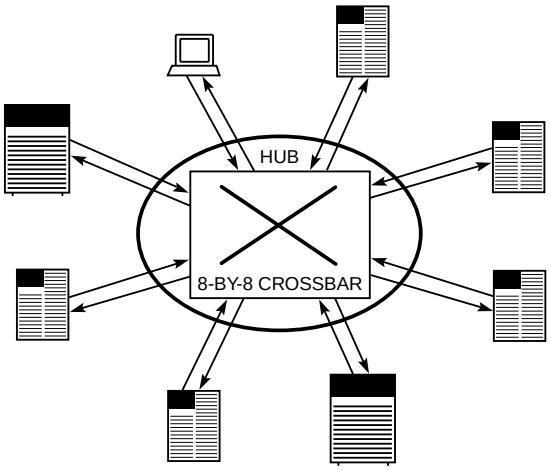
The MEMORY CHANNEL 2 link protocol has an embedded autoconfiguration mechanism that is invoked whenever a node goes on-line. The hub port and the adapter use this autoconfiguration mechanism to negotiate the mode of operation (link frequency, data path width, etc.). The same mechanism allows a two-node hubless system (a virtual hub configuration) to consistently assign node identifiers without any operator intervention or module jumpers.

MEMORY CHANNEL 2 Enhanced Software Support

MEMORY CHANNEL 2 provides four major additions to application and operating system support: (1) receive-side address remapping, (2) remote reads, (3) a global clock synchronization mechanism, and (4) conditional writes.



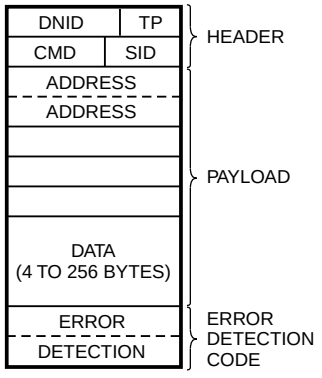
(a) MEMORY CHANNEL 1 Network



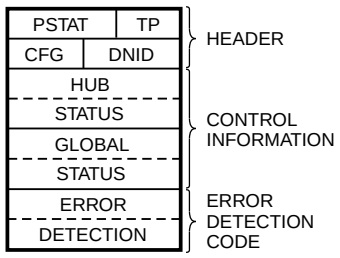
(b) MEMORY CHANNEL 2 Network

Characteristics	MEMORY CHANNEL 1	MEMORY CHANNEL 2
Channel data path width	37 bits	16 bits
Channel communication	Half duplex	Full duplex
Electrical signaling	TTL	LVDS
Optical fiber compatible	No	Yes
Link operating frequency	33 MHz	66 MHz
Peak raw data transfer rate	133 MB/s	133 + 133 MB/s
Sustained point-to-point bandwidth	66 MB/s	100 MB/s
Maximum packet size	32 bytes	256 bytes
Remote read support	No	Yes
Packet error detection	Horizontal and vertical parity	32-bit Reed-Solomon
Address space remapping	None	Receive
Supported page sizes	8 KB	4 KB and 8 KB
Hub architecture	Shared bus	Crossbar
Network bisection bandwidth	77 MB/s	800 to 1,600 MB/s

Figure 3
Comparison of First- and Second-generation MEMORY CHANNEL Architectures



(a) Data Packet



(b) Control Packet

Figure 4
MEMORY CHANNEL 2 Packet Format

On MEMORY CHANNEL 1 clusters, the network address is mapped to a local page of physical memory using remapping resources contained in the system's PCI-to-host memory bridge. All AlphaServer systems implement these remapping resources. Other systems, particularly those with 32-bit addresses, do not implement this PCI-to-host memory remapping resource. On MEMORY CHANNEL 2, software has the option to enable remapping in the receiver side of the MEMORY CHANNEL 2 adapter on a per-network-page basis. When configured for remapping, a section of the PCT is used to store the upper address bits needed to map any network page to any 32-bit address on the PCI bus. Such enhanced mapping capability will also be used to support remote access to PCI peripherals across the MEMORY CHANNEL network.

A simple remote read primitive was added to MEMORY CHANNEL 2 to support research into software-assisted shared memory. The primitive allows a node to complete a read request to another node without software intervention. It is implemented by a new remote read–on–write attribute in the receive page control table. The requesting node generates a write with the appropriate remote address (a read-request write). When the packet arrives at the receiver, its address maps in the PCT to a page marked as remote read. After remapping (if enabled), the address is converted to a PCI read command. The read data is returned as a MEMORY CHANNEL write to the same address as the original read-request write. Since read access to a page of memory in a remote node is provided by a unique network address, privileges to write or read cluster memory remain completely independent.

A global clock mechanism has been introduced to provide support for clusterwide synchronization. Global clocks, which are highly accurate, are extremely useful in many distributed applications, such as parallel databases or distributed debugging. The MEMORY CHANNEL 2 hub implements this global clock by periodically sending synchronization packets to all nodes in the cluster. The reception of such a pulse can be made to trigger an interrupt or, on future MEMORY CHANNEL–to–CPU direct-interface systems, may be used to update a local counter. The interrupt service software updates the offset between the local time and the global time. This synchronization mechanism allows a unique clusterwide time to be maintained with an accuracy equal to twice the range (max – min) of the MEMORY CHANNEL network latency, plus the interrupt service routine time.

Conditional write transactions have been introduced in MEMORY CHANNEL 2 to improve the speed of a recoverable messaging system. On MEMORY CHANNEL 1, the simplest implementation of general-purpose recoverable messaging requires a round-trip acknowledge delay to validate the message transfer, which adds to the communication latency. The MEMORY CHANNEL 2's newly introduced conditional write transaction provides a more efficient implementation that requires a single acknowledge packet, thus practically reducing the associated latency by more than a factor of two.

Memory Channel 2 Hardware

As suggested in the previous architectural description, MEMORY CHANNEL 2 hardware components are similar to those in MEMORY CHANNEL 1, namely a PCI adapter card (one per node), a cable, and a central hub.

The MEMORY CHANNEL 2 PCI Adapter Card The PCI adapter card is the hardware interface of a node to the MEMORY CHANNEL network. A block diagram of the adapter is shown in Figure 5. The adapter card is functionally partitioned into two subsystems: the PCI interface and the link interface. First in, first out (FIFO) queues are placed between the two subsystems. The PCI interface communicates with the host system, feeds the link interface with data packets to be sent, and forwards received packets on to the PCI bus. The link interface manages the link protocol and data flow: It formats data packets, generates control packets, and handles error code generation and detection. It also multiplexes the data path from the PCI format (32 bits at 33 megahertz [MHz]) to the link protocol (16 bits at 66 MHz). In addition, the link interface implements the conversion to and from LVDS signaling.

The transmit (TX) and receive (RX) data paths, both heavily pipelined, are kept completely separate from each other, and there is no resource conflict other than the PCI bus access. A special case occurs when a packet is received with the acknowledge request bit or the loopback bit set: the paths in both directions are coordinated to transmit back the response packet while still receiving the original one (employing the gray path in Figure 5). During a normal MEMORY CHANNEL 2 transaction, the transmit pipeline processes a transmit request from the PCI bus. The transmit PCT is addressed with a subset of the PCI address bits and is used to determine the intended destination of the packet and its attributes. The transmit pipeline feeds the link interface with data packets and appropriate commands through the transmit FIFO queue. The link interface formats the packets and sends them on the link cable. At the receiver, the link interface disassembles the packet in an intermediate format and stores it into the receive FIFO queue. The PCI interface performs a lookup in the

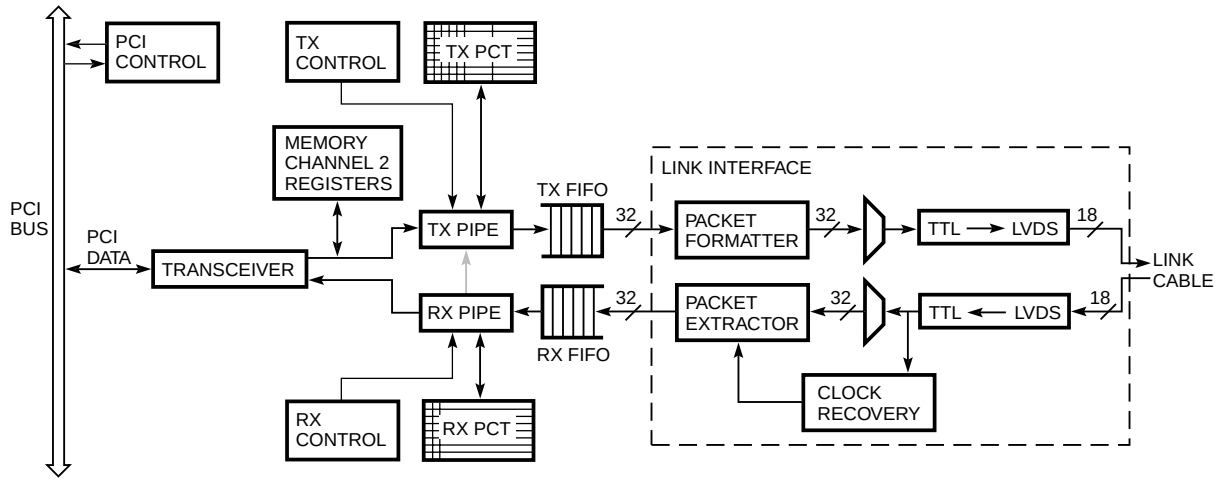


Figure 5
Block Diagram of a MEMORY CHANNEL 2 Adapter

receiver PCT to ensure that the page has been enabled for reception and to determine the local destination address.

In the simplest implementation, packets are subject to two store-and-forward delays—one on the transmit path and one on the receive path. Because of the atomicity of packets, the transmit path must wait for the last data word to be correctly taken in from the PCI bus before forwarding the packet to the link interface. The receive path experiences a delay because the error detection protocol requires the checking of the last cycle before the packet can be declared error-free. A set of control/status MEMORY CHANNEL 2 registers, addressable through the PCI, is used to set various modes of operation and to read local status of the link and global cluster status.

The MEMORY CHANNEL 2 Hub The hub is the central resource that interconnects all nodes to form a cluster. Figure 6 is a block diagram of an 8-by-8 MEMORY CHANNEL 2 hub. The hub implements a nonblocking 8-by-8 crossbar and interfaces to eight 16-bit-wide full-duplex links by means of a link interface similar to that used in the adapter. The actual crossbar has eight input ports and eight output ports, all 16 bits wide. Each output port has an 8-to-1 multiplexer, which is able to choose from one of eight input ports. Each multiplexer is controlled by a local arbiter, which is fed decoded destination requests from the eight input ports. The port arbitration is based on a fixed-priority, request-sampling algorithm. All requests that arrive within a sampling interval are considered of equal age and are serviced before any new requests. This algorithm, while not enforcing absolute arrival-time ordering among packets sent from different nodes, assures no starvation and a fair age-driven priority across sampling intervals.

When a broadcast request arrives at the hub, the otherwise independent arbiters synchronize themselves to transfer the broadcast packet. The arbiters wait for the completion of the packet currently being transferred, disable point-to-point arbitration, signal that they are ready for broadcast, and then wait for all other ports to arrive at the same synchronization point. Once all output ports are ready for broadcast, port 0 proceeds to read from the appropriate input port, and all other ports (including port 0) select the same input source. The maximum synchronization wait time, assuming no output queue blocking, is equal to the time it takes to transfer the largest size packets (256 bytes), about 4 μs, and is independent of the number of ports. As in any crossbar architecture with a single point of coherency, such broadcast operation is more costly than a point-to-point transfer. Our experience has been that some critical but relatively low-frequency operations (primarily fast locks) exploit the broadcast circuit.

MEMORY CHANNEL 2 Design Process and Physical Implementation

Figure 7 illustrates the main MEMORY CHANNEL physical components. As shown in Figure 7a, two-node clusters can be constructed by directly connecting two MEMORY CHANNEL PCI adapters and a cable. This configuration is called the virtual hub configuration. Figure 7b shows clusters interconnected by means of a hub.

The MEMORY CHANNEL adapter is implemented as a single PCI card. The hub consists of a mother-

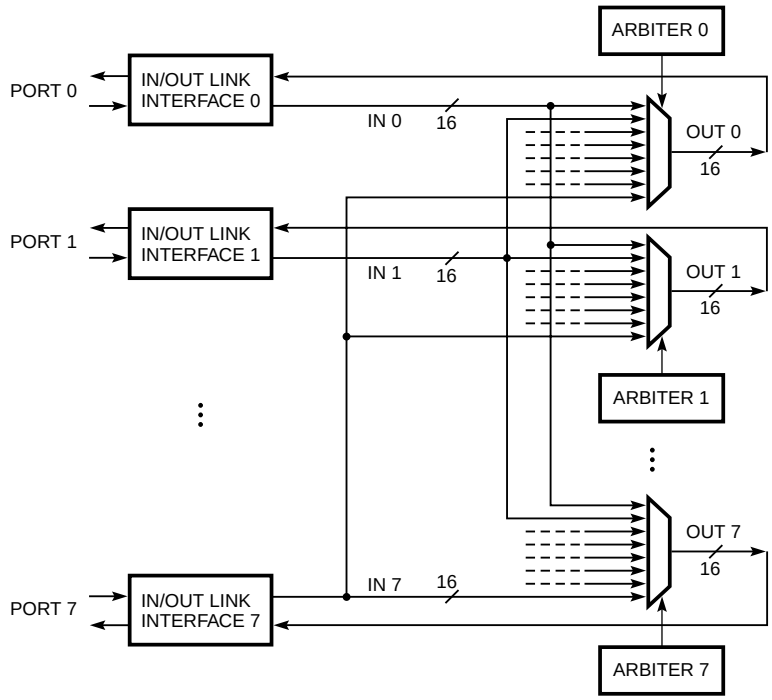
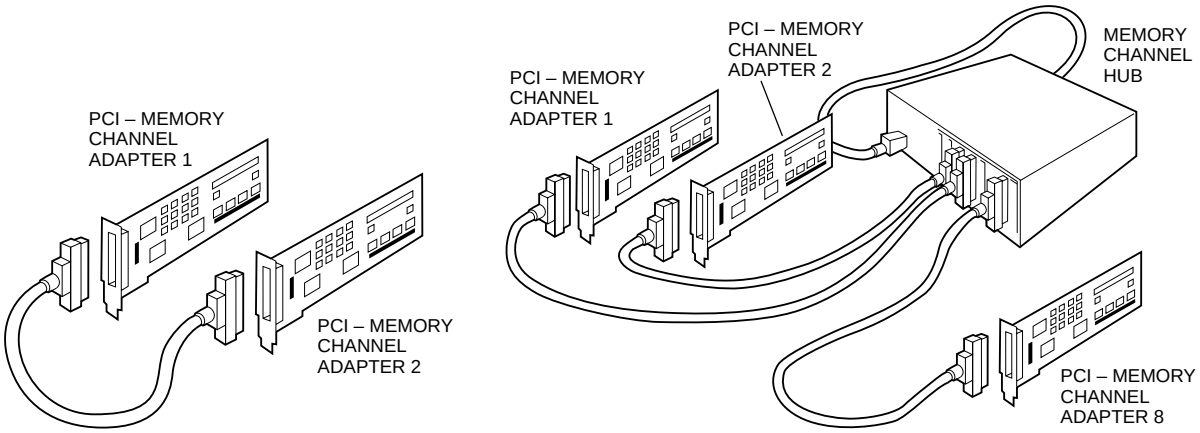


Figure 6
Block Diagram of an 8-by-8 MEMORY CHANNEL 2 Hub

board that holds the switch and a set of linecards, one per port, that provides the interface to the link cable.

The adapter and hub implementations use a combination of programmable logic devices and off-the-shelf components. This design was preferred to an application-specific integrated circuit (ASIC) implementation because of the short time-to-market requirements. In addition, some of the new functionality will evolve as software is modified to take advantage of the new features. The MEMORY CHANNEL 2 design was developed entirely in Verilog at the register transfer level (RTL). It was simulated using the Viewlogic VCS event-driven simulator and synthesized with the Synopsys tool. The resulting netlist



(a) Virtual hub mode: direct node-to-node interconnection of two PCI adapter cards

(b) Using the MEMORY CHANNEL hub to create clusters of up to 16 nodes

Figure 7
MEMORY CHANNEL Hardware Components

was fed through the appropriate vendor tools for placing and routing to the specific devices. Once the device was routed, the vendor tools provided a gate-level Verilog netlist with timing information, which was then simulated to verify the correctness of the synthesized design. Boardwide static timing analysis was run using the Viewlogic MOTIVE tool. The link interface was fitted to a single Lucent Technologies Optimized Reconfigurable Cell Array (ORCA) Series field-programmable gate array (FPGA) device. The PCI interface was implemented with one ORCA FPGA device and several high-speed AMD programmable array logic devices (PALs). Thanks to the in-system programmability of PALs and FPGAs, the MEMORY CHANNEL 2 adapter board is designed to be completely reprogrammable in the field from the system console through the PCI interface.

MEMORY CHANNEL 2 Performance

This section presents MEMORY CHANNEL 2 performance data configured in virtual hub mode (direct node-to-node connection). Wherever possible actual measured results are presented. A two-node AlphaServer 4100 5/300 cluster was used for all hardware measurements.

Network Throughput

The MEMORY CHANNEL 2 network has a raw data rate of 2 bytes every 15 ns or 133.3 MB/s. Messages are packetized by the interface into one or more MEMORY CHANNEL packets. Packets with data payloads of 4 to 256 bytes are supported. Figure 8 compares, for various packet sizes, the maximum bandwidth the MEMORY CHANNEL 2 network is capable of sustaining with the effective process-to-process bandwidth achieved using a pair of AlphaServer 4100 systems. With 256-byte packets, MEMORY CHANNEL 2 achieves 127 MB/s or about 96 percent of the raw wire bandwidth.

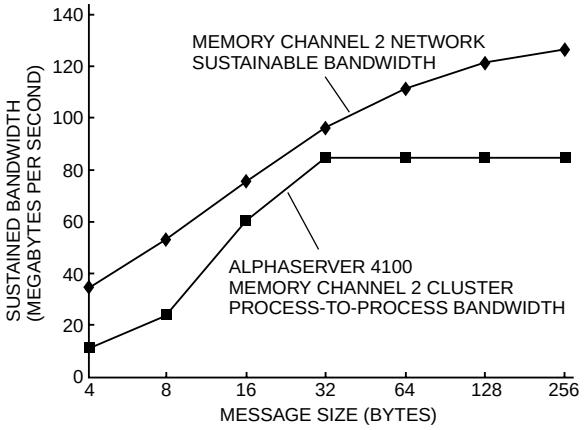


Figure 8
MEMORY CHANNEL 2 Point-to-point Bandwidth as a Function of Packet Size, Comparing Network Theoretical Limit and Sustained Process-to-process Measured Performance

For PCI writes of less than or equal to 256 bytes, the MEMORY CHANNEL 2 interface simply converts the PCI write to a similar-size MEMORY CHANNEL packet. The current design does not aggregate multiple PCI write transactions into a single MEMORY CHANNEL packet and automatically breaks PCI writes larger than 256 bytes into a sequence of 256-byte packets.

As Figure 8 shows, the bandwidth capability of the MEMORY CHANNEL 2 network exceeds the sustainable data rate of the AlphaServer 4100 system. The AlphaServer system is capable of generating 32-byte packets to the MEMORY CHANNEL 2 interface at 88 MB/s or about 10 percent less than the maximum network bandwidth at a 32-byte packet size. This represents a 33 percent bandwidth improvement over the previous-generation MEMORY CHANNEL, whose effective bandwidth was 66 MB/s. An ideal PCI host interface would achieve the full 97 MB/s, but the current AlphaServer 4100 design inserts an extra PCI stall cycle on sustained 32-byte writes to the PCI. The 32-byte packet size is a limitation of the Alpha 21164 microprocessor; future versions of the Alpha microprocessor will be able to generate larger writes to the PCI bus.

Latency

Figure 9 shows the latency contributions along a point-to-point path from a sending process on node 1 to a receiving process on node 2. Using a simple 8-byte ping-pong test, we determined that the one-way latency of this path is 2.17 μs. In the test, a user process on node 1 sends an 8-byte message to node 2. Node 2 is polling its memory waiting for the message. After node 2 sees the message, it sends a similar message back to node 1. (Node 1 started polling its memory after it sent the previous message.) One-way latency is calculated by dividing by two the time it takes to complete a ping-pong exchange. Approximately 330 ns elapse from the time a sending processor issues a store instruction until the store propagates to the sender's PCI bus. The latency from the sender's PCI to the receiver's PCI over the MEMORY CHANNEL 2 network is about 1.1 μs. Writing the main memory on the receiver node takes an additional 330 ns. Finally, the poll loop takes an average of about 400 ns to read the flag value from memory.

Table 2 shows the process-to-process one-way message latency for different types of communications

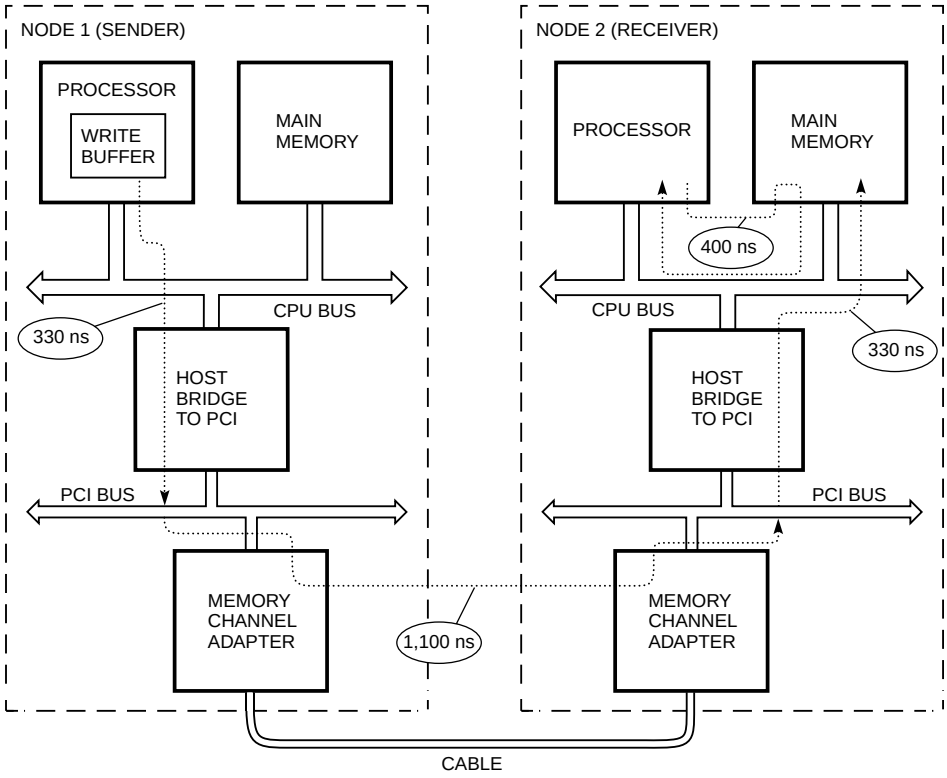


Figure 9
Latency Contributions along the Path from a Sender to a Receiver

at a fixed 8-byte message size. The first row contains the result of the ping-pong experiment previously described. For comparison, the previous generation of MEMORY CHANNEL had a ping-pong latency of 2.60 μs. The second row represents the latency for the simplest implementation of variable-length messaging. The latencies of standard communication interfaces are shown in the last two rows, namely, High Performance Fortran and Message Passing Interface. The results shown in this table are only between two and three times slower than the latencies measured for the same communication interfaces over the SMP bus of the AlphaServer 4100 system.

Table 2
MEMORY CHANNEL 2 One-way Message Latency in Virtual Hub Mode for Different Communication Interfaces

Communication Type	One-way Message Latency (Microseconds)
Ping-pong 8-byte message	2.17
8-byte message plus 8-byte flag	2.60
HPF 8-byte message	5.10
MPI 8-byte message	6.40

The latency of the MEMORY CHANNEL 2 network increases with the size of the message because of the presence of store-and-forward delays in the path. As discussed in the previous hardware description, all packets are subject to two store-and-forward delays, one on the outgoing buffer and one on the incoming buffer (required for error checking). These delays also play a role in the effective bandwidth of a stream of packets. On the one hand, smaller packets are less efficient than larger ones in term of overhead. On the other hand, smaller packets incur a shorter store-and-forward delay per packet, which can then be overlapped with the transfer of previous packets on the link, making the overall transfer more efficient. The hub performs cut-through packet routing with an additional delay of about 0.5 μs.

Summary and Future Work

This paper presents an overview of the second-generation MEMORY CHANNEL network, MEMORY CHANNEL 2. The rationale behind the major design decisions are discussed in light of the experience gained from MEMORY CHANNEL 1. A description of the MEMORY CHANNEL 2 hardware components led to the presentation of measured performance results.

Compared to other more traditional interconnection networks, MEMORY CHANNEL 1 provides unparalleled performance in terms of latency and bandwidth. MEMORY CHANNEL 2 further enhances performance by providing point-to-point bandwidth of 97 MB/s per second for 32-byte packets, an application-to-application latency of less than 2.2 microseconds, and a cross-section bandwidth of 1,000 MB/s for 8 nodes and 2,000 MB/s for 16 nodes. It also provides enhanced software support to improve the performance of the most common operations in a cluster environment, e.g., global synchronization, and reduces the complexity of the software layer by providing a more flexible address mapping. In addition, the MEMORY CHANNEL 2 network has been designed to be both hardware and software compatible with future generations on either copper or fiber-optic communication up to a distance of 3 kilometers. Future generations of the MEMORY CHANNEL architecture will benefit from the MEMORY CHANNEL 2 experience and will continue to provide enhancements to communication performance and to further refine those mechanisms introduced to support parallel cluster software.

Acknowledgments

Many thanks to Ed Benson for providing the detailed message-passing performance data contained in this paper. DIGITAL's MEMORY CHANNEL 2 team designed and implemented the second-generation MEMORY CHANNEL system described. Wayne Bortman, Robert Dickson, Marco Fillo, Richard Gillett, John Grooms, Michael McNamara, Jonathan Mooty, and Dave Pimm were the MEMORY CHANNEL 2 designers. Dale Keck, Edward Tulloch, and Ron Carn were responsible for the design verification. Brian McQuain was responsible for the printed circuit board layouts. Steve Campbell was the mechanical engineer on the hub enclosure. Special thanks go to John Grooms and Michael McNamara for their constructive comments and for proofreading the manuscript. The authors also thank the anonymous referees for their comments and suggestions, which considerably improved this paper.

References and Notes

1. G. Pfister, *In Search of Clusters: The Coming Battle in Lowly Parallel Computing* (Englewood Cliffs, N.J.: Prentice Hall, 1995).
2. M. Fillo, "Architectural Support for Scientific Applications on Multicomputers," *Series in Microelectronics,* vol. 27 (Konstanz, Germany: Hartung-Gorre Verlag, 1993).
3. D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Englewood Cliffs, N.J.: Prentice Hall, 1989).
4. J. Harris et al., "Compiling High Performance Fortran for Distributed-memory Systems," *Digital Technical Journal,* vol. 7, no. 3 (1995): 5–23.
5. R. Kaufmann and T. Reddin, "Digital's Clusters and Scientific Parallel Applications," *Proceedings of COMPCON '96,* San Jose, Calif. (February 1996).
6. M. Annaratone, C. Pommerell, and R. Ruehl, "Interprocessor Communication Speed and Performance in Distributed-Memory Parallel Processors," *Proceedings of the 16th International Symposium on Computer Architecture,* Jerusalem, Israel (May 1989).
7. C. Pommerell, M. Annaratone, and W. Fichtner, "A Set of New Mapping and Coloring Heuristics for Distributed Memory Parallel Processors," *SIAM Journal on Scientific and Statistical Computing* (January 1992).
8. S. Borkar et al., "Supporting Systolic and Memory Communication in iWarp," *Proceedings of the 17th International Symposium on Computer Architecture,* Seattle, Wash. (May 1990).
9. M. Fillo et al., "The M-Machine Multicomputer," *Proceedings of the XXVII Symposium on Microarchitecture,* Ann Arbor, Mich. (1995).
10. J. Dongarra, "Performance of Various Computers Using Standard Linear Equation Software," Technical Report CS-89-85 (Knoxville, Tenn.: University of Tennessee, Computer Science Department, December 19, 1996).
11. H. Cassanova, J. Dongarra, and W. Jiang, "The Performance of PVM on MPP Systems," Technical Report UT-CS-95-301 (Knoxville, Tenn.: University of Tennessee, Computer Science Department, 1995).
12. R. Gillett and R. Kaufmann, "Experience Using the First Generation Memory Channel for PCI Network," *Proceedings of the 4th Hot Interconnects Conference* (1996): 205–214.
13. R. Martin et al., "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," *Proceedings of the 25th International Symposium on Computer Architecture* (May 1997): 85–97.
14. T. von Eicken, D. Culler, S. Goldstein, and K. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *Proceedings of the 19th International Symposium on Computer Architecture,* Gold Coast, Australia (May 1992): 256–266.
15. K. Keeton, T. Anderson, and D. Patterson, "LogP Quantified: The Case for Low-Overhead Local Area Networks," *Proceedings of Hot Interconnects III: A Symposium on High Performance Interconnects,* Stanford, Calif. (August 1995). Also available at http://http.cs.berkeley.edu/~kkeeton/Papers/paper.html.
16. R. Gillett, "Memory Channel Network for PCI," *IEEE Micro* (February 1996): 12–18.
17. J. Brosnan, J. Lawton, and T. Reddin, "A High-Performance PVM for Alpha Clusters," *Second European PVM Conference* (1995).

18. W. Gropp and E. Lusk, "The MPI Communication Library: Its Design and a Portable Implementation," http://www.mcs.anl.gov/Papers/Lusk/mississippi/paper.html (Argonne, Ill.: Mathematics and Computer Science Division, Argonne National Laboratory).

19. W. Cardoza, F. Glover, and W. Snaman, Jr., "Design of the TruCluster Multicomputer System for the Digital UNIX Environment," *Digital Technical Journal,* vol. 8, no. 1 (1996): 5–17.

20. Information about the Transaction Processing Performance Council (TPC) is available at http://www.tpc.org.

21. M. Blumrich et al., "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *Proceedings of the 21st International Symposium on Computer Architecture* (April 1994): 142–153.

22. R. Gillett, M. Collins, and D. Pimm, "Overview of Network Memory Channel for PCI," *Proceedings of COMPCON '96,* San Jose, Calif. (1996).

23. Information about the Scalable Coherent Interface is available at http://www.SCIzzL.com.

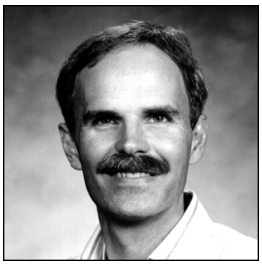
24. N. Boden et al., "Myrinet—A Gigabit-per-Second Local Area Network," *IEEE Micro,* vol. 15, no. 1 (February 1995): 29–36.

25. J. Lawton et al., "Building a High Performance Message Passing System for Memory Channel Clusters," *Digital Technical Journal,* vol. 8, no. 2 (1996): 96–116.

26. IEEE Draft Standard for Low Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI). Draft IEEE P1596.3-1995.

Biographies

Marco Fillo
Marco Fillo is a principal engineer on the MEMORY CHANNEL 2 team in the AlphaServer Engineering Group. He is responsible for the design of the MEMORY CHANNEL 2 link protocol and hub. Before joining DIGITAL in September 1995, Marco held a position as research associate at M.I.T. in the Artificial Intelligence Laboratory, where he was one of the architects of the M-Machine, an experimental multithreaded parallel computer. Marco obtained a Ph.D. in electrical engineering from the Swiss Institute of Technology, Zurich, in 1993. He is a member of the IEEE and ACM, and his research interests are parallel computer architectures and interprocessor communication networks.



Richard B. Gillett
Rick Gillett is a corporate consulting engineer in Digital Equipment Corporation's AlphaServer Engineering Group, where he designs and develops custom VLSI chips, I/O systems, and SMP systems. As DIGITAL's parallel cluster architect, he defined and led the MEMORY CHANNEL project. He holds 17 patents on inventions in SMP architectures and high-performance communication and has patents pending on the MEMORY CHANNEL for PCI network. His primary interests are high-speed local and distributed shared-memory architectures. Rick has a B.S. in electrical engineering from the University of New Hampshire. He is a member of the IEEE and the IEEE Computer Society.