MEMORY CHANNEL NETWORK FOR PCI

Richard B. Gillett

Digital Equipment Corporation

A memory-based networking approach provides clusters of computers up to 1,000 times the communication performance of conventional networks, with no compromise in cost or reliability.

By interconnecting high-volume, standard computers, system architects can construct a high-performance, modular, parallel system, called a cluster,[1,2] at a much lower cost than a specialized system. In addition, with a standard operating system running on each computer, or node, a cluster can cleanly survive the failure of an individual computer, disk, network, or other subsystem. A cluster can also survive many types of software and system management failures, which often dominate the failure rate of modern computer systems.

Due to dramatic improvements in microprocessor performance, clusters of symmetric multiprocessors (SMPs) now equal or exceed the performance of the largest mainframes and special-purpose parallel systems. A conventional network (Ethernet, FDDI, ATM, Fiberchannel) would seem to be the natural technology to interconnect these high-powered SMP systems. Unfortunately, conventional-network performance has not kept pace with microprocessor performance. This is particularly true of two key performance attributes, message latency and overhead, both of which have received much less attention than bandwidth in conventional network design. (The old adage that thunder is impressive, but lightning does the work is relevant here. All too often bandwidth is impressive, but overhead and latency do the work.) As a result, interconnection performance often limits a cluster's parallel performance.

Digital Equipment Corporation has designed a new network, Memory Channel for PCI (MC for short), to address the unique needs of clusters. MC implements cluster-wide virtual shared memory. Real-world measurements confirm that it reduces overhead and latency by two to three orders of magnitude, while clusters retain the ability to survive any single failure. MC supports any computer using the industry-standard PCI (peripheral-component interconnect) bus, and system builders can implement the network at very low cost. Unlike standard networks, MC's performance depends almost totally on semiconductor technology, so it should improve at a rate similar to that of microprocessor hardware performance.

Standard-network performance

A standard network's intersystem communication overhead limits a cluster's ability to achieve its full potential as a parallel and available system. Figure 1 shows communication performance across the memory hierarchy of a modern SMP system (the AlphaServer 8400, also known as the Turbolaser) connected to a conventional network. The graph illustrates three key aspects of communication performance for each memory level:

- bandwidth: maximum sustained data rate;
- latency: total elapsed time from initiation of a communication operation until it completes; and
- overhead: CPU time consumed by a communication operation.

For all levels except the network, we measured performance for a read (load) operation. For the standard-network level, we used a one-way, process-to-process message operation, using the TCP/IP network protocol standard for Unix over an FDDI network. The message size was 32 bytes. We measured one-way message latency by performing a round-trip message test between two nodes and then dividing the overhead and latency results by two.

The graph shows the expected trade-offs among size, distance, and speed. Increased storage capacity comes at the expense of lower bandwidth and higher latency and

0772-1732/96/$5.00 © 1996 IEEE

overhead. For example, bandwidth falls at a relatively consistent rate from over 20,000 Mbytes per second in the register file to 300 Mbytes/s in shared memory. This trend ends when communication leaves the SMP domain and enters the network. There, bandwidth falls by a factor of 30, or about five times the largest drop in the higher levels.

Latency and overhead increase consistently from 3 ns and 1 ns, respectively, in the register file to about 250 ns in shared memory. As it was for bandwidth, the jump from shared memory to network is much larger. Sending a message over a network requires more than 1,000 times the latency and overhead of shared memory. This is the crux of the problem with standard networks and clusters.

Although some real-world applications run well with little communication between cluster nodes, many cannot. A performance example presented later is a real-world application that requires a large amount of communication. In that application, a molecular-modeling program running across two cluster nodes on a standard network uses 75 percent of each node's processor time for communication, leaving only 25 percent for useful work. As a result, twice as much hardware does half as much work.

Based on the rough trends established in the lower levels of the hierarchy, cluster communication should have an overhead and latency of about 10 times 250 ns, or 2.5 µs. Cluster bandwidth should be about 100 Mbytes/s. The MC project's main goal was to approach this level of performance. But before I explain how MC reduces overhead, let's briefly examine why standard networks have such high overhead and latency.

Network communication overhead is only slightly smaller than latency. This indicates that most latency is the direct result of overhead, which is processor time spent on work associated with communication. This conclusion is consistent with performance studies showing that high-performance network hardware directly contributes only a small portion of total latency.[3]

So the problem is the processor time required for network communication. The network is the first memory level in which the application does not directly perform communication; instead, the application calls an operating system service. The operating system service that provides network communication is complex, with three primary layers: application interface, network protocol, and network driver. Much has been written about communication performance and why network service path lengths are so high. Opinions have polarized over which layer is to blame. In reality, all three add significant overhead. In a typical implementation, overhead spreads almost equally across the layers.

Two other observations increase the importance of reducing network communication overhead. First, historical data show that performance of operating system services improves less than application performance as microprocessors speed up. This means communication overhead is growing as a percentage of total execution time. With a one-way message overhead of 250 µs, transmitting and receiving a single 32-byte message often consumes the same processor time as 150,000 application instructions.

Second, cluster traffic tends to be dominated by small messages—the worst traffic pattern for a communication subsystem with very high overhead compared to bandwidth. To see this issue in perspective, assume that messages smaller than 200 bytes dominate typical cluster traffic. With one-way message overhead of 250 µs and a 200-byte message size, a dedicated processor can transmit 4,000 and receive 4,000 messages per second. This corresponds to a network bandwidth of only 1.6 Mbytes/s, which two 10-Mbit/s Ethernets could support.

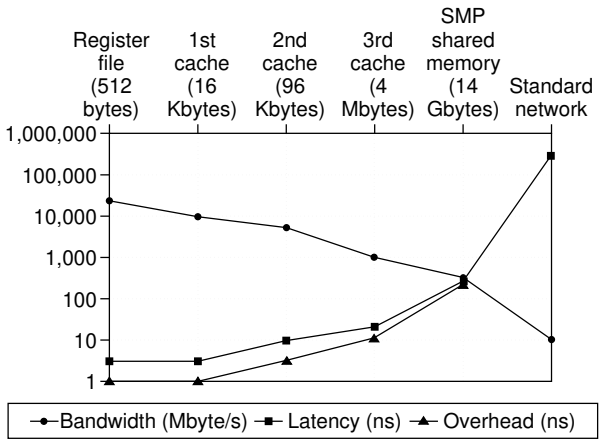


Figure 1. Digital AlphaServer 8400 memory hierarchy communication characteristics (SMP: symmetric multiprocessor system).

Digital's research[4] shows that improved software implementations can push the overhead and latency of a one-way message over a modern network (like ATM) to 65 µs. While this would be a tremendous improvement, it is still more than 20 times the desirable cluster communication latency derived earlier. In addition, the fastest implementations still have potential error recovery problems.

Requirements and design foundation

Many requirements of long-haul, heterogeneous networks impede implementation of low-overhead interconnections. Clusters are a constrained subset of networked systems. The following key constraints differentiate clusters from general networked systems:

- Clusters are homogeneous, with only one operating system type.
- Cluster node count is small—typically two to eight nodes (two to 96 processors).
- Cluster nodes are usually physically close and often physically secure (in the same computer room or wiring closet, for example).
- Since they are not limited to industry-standard protocols such as TCP/IP, we can tailor cluster communications for optimized networks.

By exploiting these natural cluster constraints, we can design a network with performance characteristics suited for

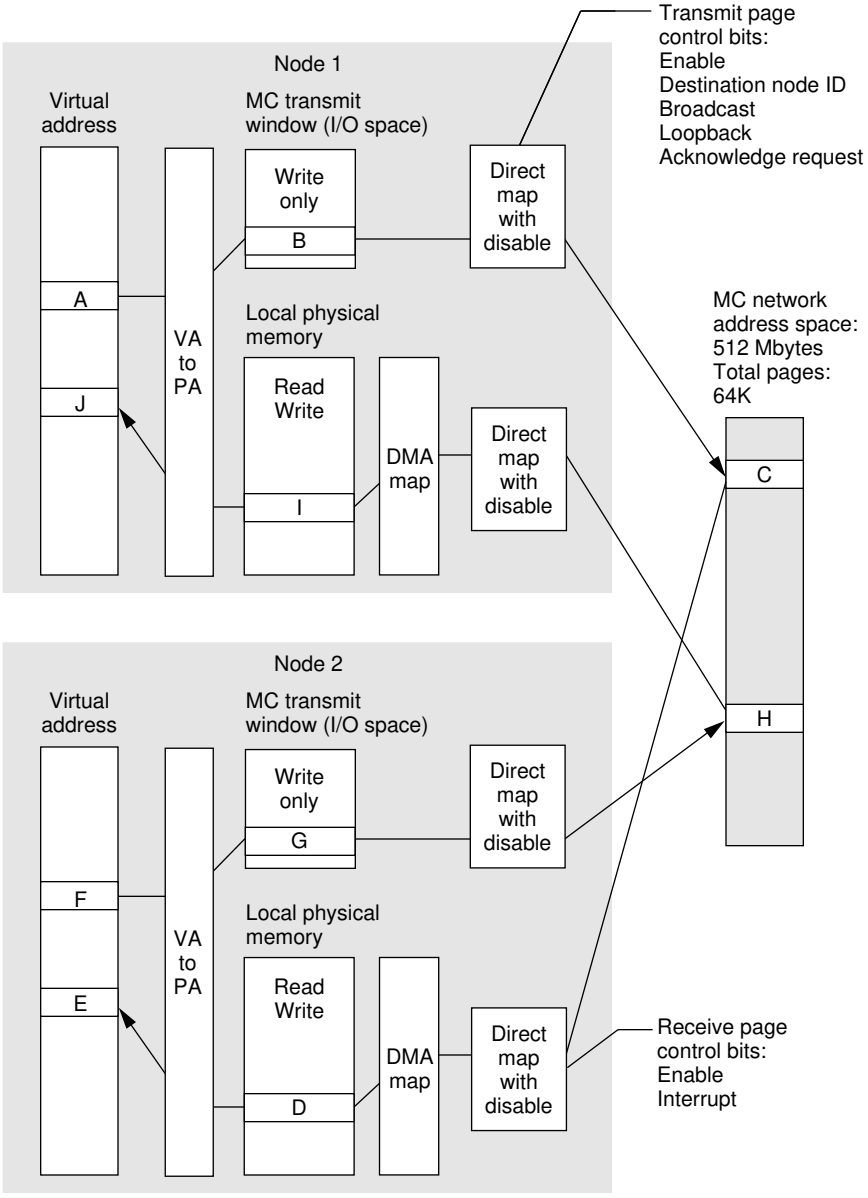


Figure 2. MC's virtual-shared-memory implementation (letters A through J: pages; VA to PA: virtual address to physical address).

clusters. But unless the performance gains are great, developing such a network would not be worth the effort after the industry's huge investment in standard networks. Therefore, we had to define aggressive goals to justify our own investment:

- Drop cluster communication latency and overhead by factors of 100 and 1,000 respectively, thus achieving one-way message latency of less than 5 µs and overhead of less than 0.5 µs.
- Maintain the ability to repair hardware while the cluster is running (hot-swap capability).
- Support full detection and recovery (including fallback to a backup network) from network communication errors (the SMP "crash-on-error" model is unacceptable).
- Connect to the industry-standard PCI bus for compatibility with all PCI-based Alpha and Intel computers.

We could achieve these goals only by radically changing the network hardware-software interface. Clearly, to support message latency of less than 5 µs, the message software must execute entirely at user level (a system call alone can consume more than 5 µs). Approaching this network problem from the SMP world, we looked for a way to modify the SMP memory model to support the needs of clusters. If we could map the network into the virtual-address space of the individual nodes, direct user access would be straightforward. Standard techniques for protecting virtual-address space would isolate and protect users from each other.

We chose Encore Computer Corporation's reflective memory technology as the foundation for the new network. (Encore has trademarked the name Memory Channel for its implementation of a reflective memory network. Digital has acquired the right to use the name for its PCI bus implementation of the network.) In its simplest form, reflective memory consists of interconnected dual-port memory cards residing in each node's I/O space and comprising a simple distributed-memory system. When a node writes to its network memory card's local port, all the other nodes' network memory cards store copies of the write.

After addressing the performance requirements, our next design challenge was to ensure that we could layer a very efficient, reliable, recoverable communication subsystem on top of the very fast hardware. We accomplished this by giving MC's adapter hardware a large part of the responsibility for high-level communication functions such as error handling and flow control. In particular, MC's hardware provides

- automatic, hardware-based error detection on all message writes;
- extremely low (essentially zero) transmission error rates;
- strict message-write ordering even under errors;

- full hardware-based flow control;
- a simple, flexible hardware primitive to determine if a message reached all destinations free of error;
- a fast hardware lock primitive;
- partition-proof communication behavior (Partitioning is a communication network failure that splits the cluster into two clusters, each convinced the other's nodes have failed. Standard networks are susceptible to partitioning.);
- single-level crossbar switch architecture;
- full hot-swap support;
- hardware-enforced cluster behavior (including heartbeat time-out and flow control time-out); and
- complete support for broadcast, multicast, and point-to-point circuits.

Moreover, the hardware imposes no special PCI bus requirements.

Architecture and implementation

The MC architecture sits comfortably between SMP architectures on one side and standard networks on the other. It uses shared memory to provide high-performance communication and extends the traditional responsibility of network hardware to provide high availability. As described earlier, reflective memory is the foundation of the MC network. The commercial implementation modifies and extends the reflective-memory concept.

MC implements a form of distributed shared memory. Figure 2 shows how MC connects the virtual-address spaces of two cluster nodes. The basic network primitive is a memory-mapped circuit that provides a write-only connection between a page of virtual-address space on a transmitting node and a page of physical memory on a receiving node. Through this connection, applications can send information directly to other nodes, using standard store instructions with no operating system intervention. These connections support any write operation on memory (including all forms of byte masking). In the current implementation, they do not directly support read operations.

Figure 2 shows two write-only MC connections. The first connects a page of virtual-address space in node 1 (page A) to a page of virtual-address space in node 2 (page E). To establish this connection, the application calls a cluster service, which allocates a page of MC address space (page C). Each node then uses its MC adapter and local node DMA maps to establish a mapping between that MC address space and the corresponding local virtual-address space. In node 1, the virtual-to-physical map points to a page in node 1's MC transmit window (page B). The transmit window maps directly to the MC network address space. Connecting pages B and C requires enabling the corresponding page for transmit in the MC adapter's page control table (PCT). The PCT contains a direct-mapped entry for every page in the MC network address space. The following bits control the nature of the memory-mapped circuit:

- enable for transmit;
- enable for receive;
- broadcast, multicast, or point-to-point;
- make local copy;
- autoacknowledge writes at all receivers (hardware-based acknowledge); and
- interrupt host on write.

Node 2 establishes a connection between pages C and D by enabling the corresponding PCT entry for receive and by setting up the system I/O page table, which maps I/O from PCI to host physical memory. Finally, node 2 establishes a physical-to-virtual mapping between page D and page E. Note that page D must be locked down by the operating system; swapping out page D would effectively break the MC connection.

MC uses shared memory to provide high-performance communication and extends the traditional responsibility of network hardware to provide high availability.

After the cluster establishes this circuit, a node 1 process with write privileges for page A can write the contents of pages D and E directly to node 2. Node 1 has no physical-memory storage associated with page A or B. The only storage is in node 2 (physical page D). MC has essentially "stretched" a write port to page D across the network and attached it to page A in node 1. The cluster establishes the second write-only connection in Figure 2, from page F in node 2 to page J in node 1, in the same way.

MC has page-level connection granularity, which is 8 Kbytes for Alpha systems. The current hardware supports 64K connections across a 512-Mbyte MC address space (64K × 8 Kbytes). Connections can be point-to-point, multicast, or broadcast. Only a clusterwide service can establish connections, ensuring protection and security and allowing each node to control if, when, and where it exposes its local address space to the MC address space. This isolation of local address spaces is the basis for recovery from node failures; a node failure affects only a portion of the node's local address space.

Once connections are established, system or user processes authorized to access the required virtual pages can use them. The overhead of establishing a connection is much higher than the overhead of using the connection. This page-level connection approach is similar to the circuit-switching technology used in the telephone system. It is this approach that makes MC so much faster than a conventional network. Conventional networks lack sufficient stored information to map connections directly into the virtual-address space of a

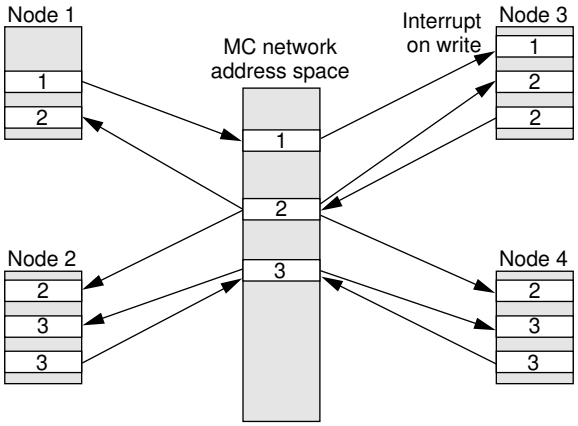


Figure 3. Examples of MC connections (numerals 1 through 3 represent connections).

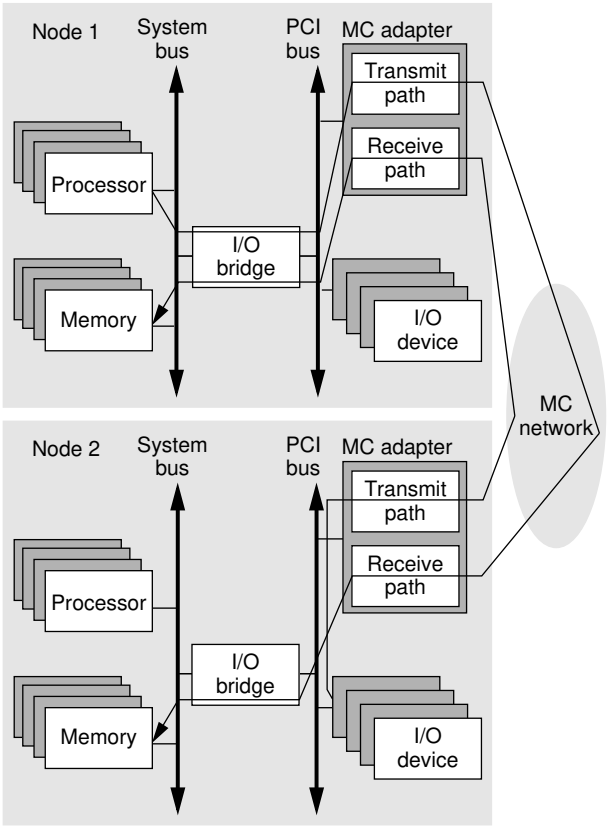


Figure 4. Hardware block diagram of a two-node MC cluster.

user process and therefore must perform checks on every message transmission. MC performs these checks only at circuit setup time, thus amortizing the checking overhead across every message sent over a connection.

Through creative use of the PCT, a cluster establishes various connections. Figure 3 shows the following examples:

- connection 1—unidirectional (written by node 1), point-to-point (received by node 3) with interrupt when written;
- connection 2—unidirectional (written by node 3), broadcast (received by all nodes); and
- connection 3—bidirectional (written by nodes 2 and 4), multicast (received by nodes 2 and 4).

Focusing on hardware, Figure 4 shows the interconnections of processors, memories, I/O devices, and MC. The MC adapter is a single-slot PCI option. On the PCI bus, the adapter's transmit path appears as a naturally aligned, 512-Mbyte, write-only memory region (which can be positioned anywhere in the 4-G byte PCI address space). Its receive path maps to a separate 512-Mbyte region on the PCI.

Figure 4 shows a path from a processor in node 1 to memory in node 2. Let's examine how a 32-byte write makes its way down this path. Assuming an Alpha processor, the program will perform consecutive write (store) instructions to the desired offset in the MC transmit window (Figure 2). The MC transmit window is in I/O space and therefore bypasses all caches, but the writes merge in one of Alpha's 32-byte write buffers.

When the processor flushes the write buffer, a full 32-byte write crosses the system bus and transfers to the PCI bus. This write selects the MC adapter, which uses information from the PCT to encapsulate the PCI write into an MC packet. The MC packet consists of the PCI write with a header identifying the destination and a trailer containing a 32-bit CRC (cyclic redundancy code). The adapter on node 2 receives this write, strips off the MC header and trailer, and sends it to the PCI bus. Node 2's I/O bridge maps the write to the proper physical-memory page, just as it would for any normal I/O operation. A process on node 2 can fully cache the contents of pages written by MC because Alpha and Intel systems both support full processor cache coherency for all I/O operations.

Figure 4 also shows a path from an I/O device on node 2 to memory on node 1. One of the MC architecture's powerful capabilities is that standard I/O devices can directly use MC connections just as processors can. This capability is particularly useful because MC supports the PCI's full byte-mask capabilities; thus, I/O devices can perform transfers exactly as if the memory were local. This capability allows a high-performance implementation of I/O shipping. I/O shipping, part of a Digital Unix service called distributed raw disk, gives clusterwide disk access even to devices connected to only one node. Using direct I/O to MC, the serving node does not touch the data from disk; instead, the data go directly to their final destination in the requesting node.

Performance results

We compared MC's performance with that of a conventional network (FDDI) and with direct interprocessor communication in an SMP system. We quantified performance for process-to-process message passing, a streamlined write-and-run message protocol especially suitable for MC, and a real-world application.

Process-to-process message passing. Figure 5 shows the message-passing performance of three hardware inter-

connects (FDDI, MC, and SMP) using three programming interfaces:

- *PVM*: message passing using a complete implementation of the industry-standard Parallel Virtual Machine application programming interface from Oak Ridge National Laboratories;
- *HPF*: message passing automatically generated by Digital's High-Performance Fortran compiler, using the HPF directives for parallel applications; and
- *direct*: explicit, user-written message passing.

In all cases, the test programs implemented a simple round-trip message loop (message size was 32 bytes), which was repeated to ensure an accurate measurement. We calculated the one-way message latency numbers by dividing the round-trip time by two.

The hardware environment was a two-node AlphaServer 2100 4/200 cluster, connected with the highest performance PCI-FDDI adapter and an MC connection. The FDDI tests all used the TCP/IP network protocol. The MC and SMP tests used optimized communication libraries.

Compared to the FDDI network, MC reduced message-passing latency for the standard interfaces (PVM and HPF) by a factor of about 25. The direct interface took better advantage of MC and reduced latency by a factor of 83. Compared to MC, the SMP system reduced latency by additional factors of about 2.5 for the standard interfaces and 5 for the direct interface. For the direct interface, most of the SMP's latency advantage over MC is due to a decrease of about 3 µs in write latency in its bus hardware.

We obtained the PVM and HPF results using a service that supports complete recovery from transmission errors without application involvement. By dropping transparent recovery, we could still implement a fully protected messaging system (that is, the receiver would never act on a bad message), but we would have to handle errors at another application level. Since the MC error rate is essentially zero, dropping transparent recovery may be a practical solution for all implementations except core-cluster services, which transparently switch to another MC network on a hardware failure. Without transparent recovery, PVM and HPF latency should drop from 20 µs to 15 µs (the difference is the cost of MC's remote-acknowledge operation). Overhead should decrease much more, since messages can then be write-and-run operations. In a write-and-run operation, the transmitter performs the write into a fast write buffer; the processor is immediately free to perform other work. Meanwhile, the buffer initiates the transfer to the MC network. The next section quantifies the performance benefit of this type of message write.

Write-and-run message overhead. For a standard network, we have shown that overhead is a large percentage of latency. MC naturally supports the ability to perform write-and-run messaging, in which message overhead can be much smaller than message latency. To measure message overhead, we ran a simple program that generates a fixed number of MC messages at a specified rate. We calculated overhead by measuring the runtime reduction when the same program ran with the actual MC message writes removed. We also measured the effectiveness of deferred write buffering at various message-sending rates.

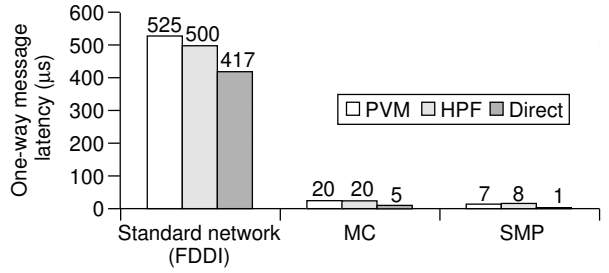


Figure 5. One-way, process-to-process message latency.

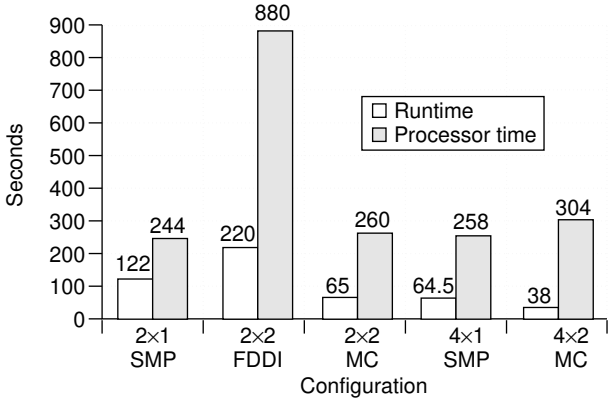


Figure 6. Real-world (PVM) application performance.

For small messages, sent at data rates significantly less than the network's maximum rate, MC's measured overhead was approximately 150 ns—2,500 times lower than FDDI's overhead. This result shows that MC and similar networks can provide overheads much smaller than latency (the ratio in this case was 30:1). As we expected, as the data rate increased, the overhead approached the value calculated by dividing message size by the maximum sustained data rate.

Application performance. We have shown that MC speeds up communication by factors of 25 to 83. Figure 6 shows the resulting performance improvement of an unmodified real-world application, a parallel molecular-modeling program that uses PVM. The figure shows the application's end-to-end runtime and total processor time for various cluster configurations.

The application ran for 122 seconds, consuming a total of 244 processor seconds, on a two-processor, single-node (2×1) SMP configuration. High communication overhead resulted in much lower performance on a FDDI-connected 2×2 cluster, where runtime increased by a factor of two, and total processor time jumped by a factor of four. Replacing the network with MC dropped total processor time almost to the single-node value. With much lower communication overhead, the four processors almost halved the original runtime to 65 sec-

The hardware's simple and powerful communication model supports error handling at almost no cost or complexity to the application; guaranteed ordering under errors is the key innovation.

onds. We obtained essentially the same results with four processors connected on a single SMP bus. This indicates that the increased memory bandwidth of two nodes (compared to one) balances MC's greater overhead (compared to SMP). The final configuration had four processors in each of two nodes, and runtime dropped by another 42 percent.

THE MEMORY CHANNEL FOR PCI'S performance gains are the result of a system design approach that exploits natural cluster constraints to define a memory-based network. MC implements a form of virtual shared memory that permits applications to completely bypass the operating system and perform cluster communication directly from the user level. The hardware's simple and powerful communication model supports error handling at almost no cost or complexity to the application; guaranteed ordering under errors is the key innovation. The end result: Real-world cluster communication latency dropped by up to two orders of magnitude, and overhead by up to three orders of magnitude. These improvements elevate a "lowly" set of standard PCI computers running Unix into an impressive, highly available, parallel computing system. (Pfister[2] affectionately refers to clusters as "lowly parallel computing" because they represent a parallel-system technology based on components from the low end of the computer market.)

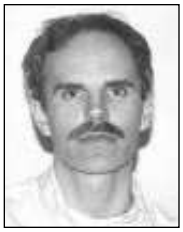
The first-generation Memory Channel for PCI began shipping early this year. Our future plans include bandwidth increases through an optimized crossbar switch and PCI adapter cards. We will increase cable length and bandwidth by moving to low-voltage differential signaling and an optical technology. These changes will significantly decrease process-to-process latency and overhead. Finally, to support continuing research on software-assisted coherent shared memory in MC clusters, we will add a read operation to the network.

Acknowledgments

Many people played critical roles in bringing this technology to market. Four teams deserve special recognition: The Digital/Encore MC team developed the first hardware and software prototypes. Digital's Unix cluster team implemented the software environment that takes advantage of the MC interconnect. Digital's high-performance computing group implemented the optimized application interfaces (including PVM). Finally, Digital's MC for PCI hardware team developed the entire hardware interconnect described in this article.

References

1. N. Kronenberg, H. Levy, and W. Strecker, "VAXClusters: A Closely Coupled Distributed System," *ACM Trans. Computer Systems*, Vol. 4, No. 2, May 1986, pp. 130-146.
2. G. Pfister, *In Search of Parallel Clusters: The Coming Battle in Lowly Parallel Computing*, Prentice-Hall, Upper Saddle River, N.J., 1995.
3. K. Keeton, T.E. Anderson, and D.A. Patterson, "LogP Quantified: The Case for Low-Overhead Local Area Networks," *Hot Interconnects III Symp. Record*, 1995; http://http.cs.berkeley.edu/~kkeeton/Papers/papers.html.
4. D. Scales, M. Burrows, and C. Thekkath, "Experiences with Parallel Computing on the AN2 Network," to be published in *Proc. 10th IEEE Int'l Parallel Processing Symp.*, IEEE Computer Society Press, Los Alamitos, Calif., 1996.



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

Send comments and questions about this article to Richard B. Gillett, AlphaServer Development, Digital Equipment Corp., PKO3-1/R50, 129 Parker St., Maynard, MA 01754; gillett@eng.pko.dec.com.

Reader Interest Survey

Indicate your interest in this article by cicling the appropriate number on the Reader Service Card.

Low 153 Medium 154 High 155