THE QUADRICS NETWORK: HIGH-PERFORMANCE CLUSTERING TECHNOLOGY

THE QUADRICS NETWORK EXTENDS THE NATIVE OPERATING SYSTEM IN PROCESSING NODES WITH A NETWORK OPERATING SYSTEM AND SPECIALIZED HARDWARE SUPPORT IN THE NETWORK INTERFACE. DOING SO INTEGRATES INDIVIDUAL NODE'S ADDRESS SPACES INTO A SINGLE, GLOBAL, VIRTUAL-ADDRESS SPACE AND PROVIDES NETWORK FAULT TOLERANCE.

Fabrizio Petrini
Wu-chun Feng
Adolfy Hoisie
Salvador Coll
Eitan Frachtenberg
Los Alamos National Laboratory

The interconnection network and its associated software libraries are critical components for high-performance cluster computers and supercomputers, Web-server farms, and network-attached storage. Such components will greatly impact the design, architecture, and use of future systems.

Key solutions in high-speed interconnects include Gigabit Ethernet,[1] GigaNet,[2] the Scalable Coherent Interface (SCI),[3] Myrinet,[4] and the Gigabyte System Network (Hippi-6400).[5] These interconnects differ from one another in their architecture, programmability, scalability, performance, and ability to integrate into large-scale systems. While Gigabit Ethernet resides at the low end of the performance spectrum, it provides a low-cost solution. GigaNet, SCI, Myrinet, and the Gigabyte System Network provide programmability and performance by adding communication processors on the network interface cards and implementing different types of user-level communication protocols.

The Quadrics network (QsNet) surpasses these interconnects in functionality with an approach that integrates a node's local virtual memory into a globally shared, virtual-memory space; provides a programmable processor in the network interface that allows the implementation of intelligent communication protocols; and delivers integrated network fault detection and fault tolerance. Consequently, QsNet already possesses many of the salient aspects of InfiniBand,[6] an evolving standard that also gives an integrated approach to high-performance communication.

QsNet

QsNet consists of two hardware building blocks: a programmable network interface called Elan and a high-bandwidth, low-latency communication switch called Elite.[7] Elite switches can be interconnected in a fat-tree topology.[8] With respect to software, QsNet provides several layers of communication libraries that trade off between performance and ease of use. QsNet combines these hardware and software components to implement efficient and protected access to a global virtual memory via remote direct memory access (DMA) operations. It also enhances network fault tolerance via link-level and end-

0272-1732/02/$17.00 © 2002 IEEE

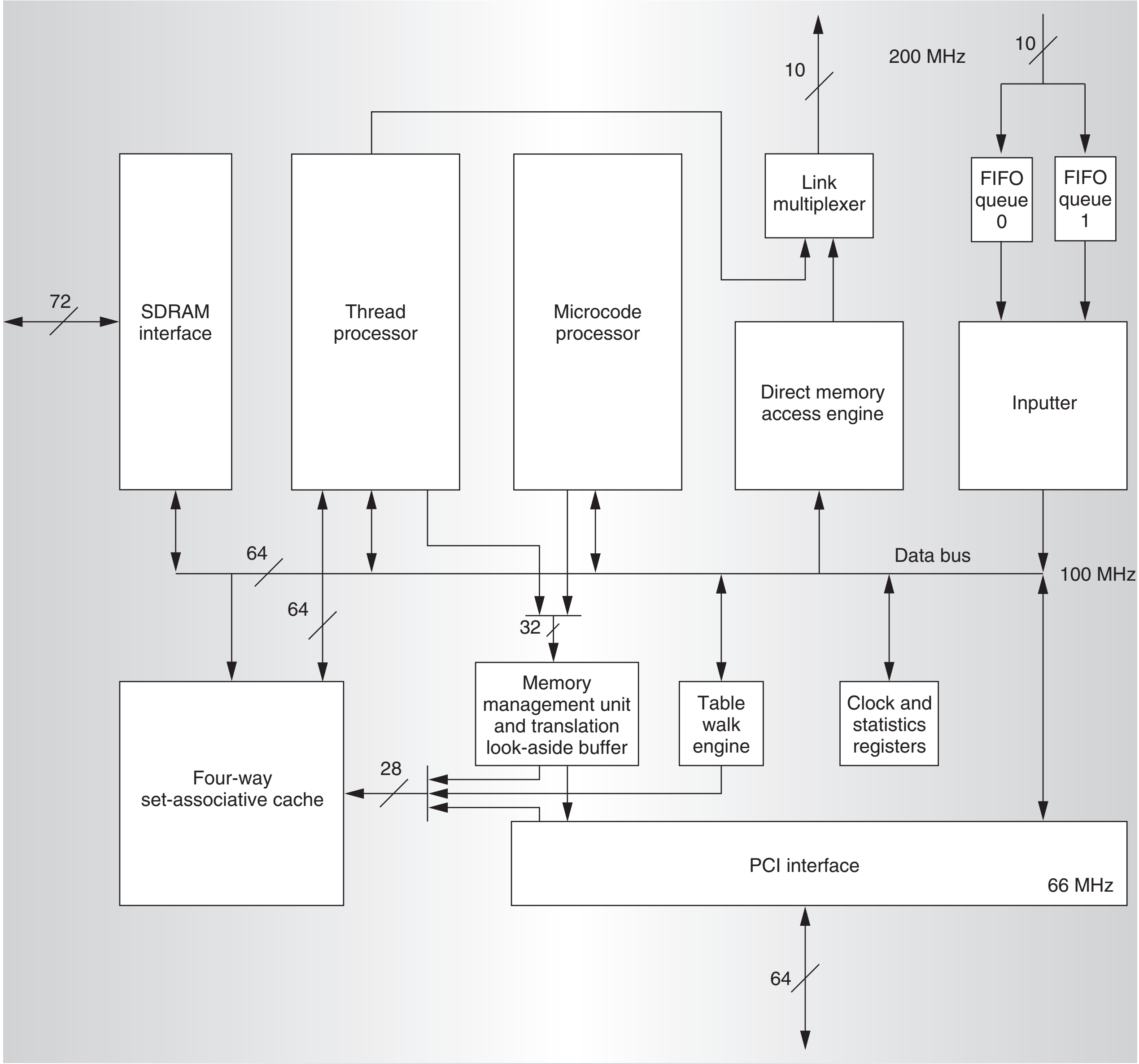

Figure 1. Elan functional units.

to-end protocols that detect faults and automatically retransmit packets.

Elan network interface

The Elan network interface (we refer to the Elan3 version of Elan in this article) connects the Quadrics network to a processing node containing one or more CPUs. In addition to generating and accepting packets to and from the network, Elan provides substantial local processing power to implement high-level, message-passing protocols such as the Message-Passing Interface (MPI). The internal functional structure of Elan, shown in Figure 1, centers around two primary processing engines: the microcode processor and the thread processor.

The 32-bit microcode processor supports four hardware threads. Each thread can independently issue pipelined memory requests to the memory system. Up to eight requests can be outstanding at any given time. Scheduling for the microcode processor permits a thread to wake up, schedule a new memory access based on the result of a previous memory access, and go back to sleep in as few as two system clock cycles.

The four microcode threads are for

- the inputter, which handles input transactions from the network;
- the DMA engine, which generates DMA packets to write to the network, prioritizes outstanding DMAs, and time-slices large DMAs to prevent adverse blocking of small DMAs;
- processor scheduling, which prioritizes and controls the thread processor's scheduling and descheduling; and
- the command processing, which handles requested operations (commands) from the host processor at the user level.

The thread processor is a 32-bit RISC processor that helps implement higher-level messaging libraries without explicit intervention from the main CPU. To better support the implementation of high-level message-passing libraries without the main CPU's explicit intervention, QsNet augments the instruction set with extra instructions. These extra instructions help construct network packets, manipulate events, efficiently schedule threads, and block save and restore a thread's state when scheduling.

The memory management unit (MMU) translates 32-bit virtual addresses into either 28-bit local SDRAM physical addresses or 48-bit peripheral component interconnect (PCI) physical addresses. To translate these addresses, the MMU contains a 16-entry, fully associative, translation look-aside buffer, and a small data path and state machine to perform table walks to fill the translation look-aside buffer and save trap information when the MMU experiences a fault.

Elan contains routing tables that translate every virtual processor number into a sequence of tags that determine the network route. The system software can load several routing tables to provide different routing strategies.

Elan has an 8-Kbyte memory cache (organized as four sets of 2 Kbytes) and a 64-Mbyte SDRAM. The cache line size is 32 bytes. The cache performs pipelined fills from SDRAM and can issue multiple cache fills and write backs for different units while still servicing accesses for units that hit on the cache. The SDRAM interface is 64 bits in length with eight check bits added to provide error-correcting code. The memory interface also contains 32-byte write and read buffers.

The link logic transmits and receives data from the network and generates 9 bits and a clock signal on each half of the clock cycle. Each link provides buffer space for two virtual channels with a 128-entry, 16-bit FIFO RAM for flow control.

Elite switch

Elite provides

- eight bidirectional links supporting two virtual channels in each direction,
- an internal 16 × 8 full crossbar switch (the crossbar has two input ports for each input link to accommodate two virtual channels),
- a nominal transmission bandwidth of 400 Mbytes/s in each link direction and a flow-through latency of 35 ns,
- packet error detection and recovery with cyclic-redundancy-check-protected routing and data transactions,
- two priority levels combined with an aging mechanism to ensure fair delivery of packets in the same priority level,
- hardware support for broadcasts, and
- adaptive routing.

QsNet connects Elite switches in a quaternary fat-tree topology, which belongs to the more general class of k-ary n-trees.[9] A quaternary fat tree of dimension n is composed of 4^n processing nodes and $n \times 4^{n-1}$ switches interconnected as a delta network; it can be recursively built by connecting four quaternary fat trees of dimension $n - 1$. Figure 2 shows quaternary fat trees of dimensions 1, 2, and 3.

Packet routing. Elite networks are source routed. The Elan network interface, which resides in the network node, attaches route information to the packet header before injecting the packet into the network. The route information is a sequence of Elite link tags. As the packet moves inside the network, each Elite switch removes the first route tag from the header and forwards the packet to the next Elite switch in the route or to the final destination. The routing tag can identify either a single output link or a group of links.

The Elan interface pipelines each packet transmission into the network using wormhole flow control. At the link level, the Elan interface partitions each packet into smaller 16-bit units called flow control digits or *flits*.[10] Every packet closes with an end-of-packet token, but the source Elan normally only sends the end-of-packet token after receipt of a packet acknowledgment token. This process implies that every packet transmission creates a virtual circuit between source and destination.

Network nodes can send packets to multiple destinations using the network's broadcast capability.[11] For successful broadcast packet delivery, the source node must receive a positive acknowledgment from all the broadcast group recipients. All Elan interfaces connected to the network can receive the broadcast packet but, if desired, the sender can limit the broadcast set to a subset of physically contiguous Elans.

Global virtual memory

Elan can transfer information directly between the address spaces of groups of cooperating processes while maintaining hardware protection between these process groups. This capability—called *virtual operation*—is a sophisticated extension to the conventional virtual memory mechanism that is based on two concepts: Elan virtual memory and Elan context.

Elan virtual memory. Elan contains an MMU to translate the virtual memory addresses issued by the various on-chip functional units (thread processor, DMA engine, and so on) into physical addresses. These physical memory addresses can refer to either Elan local memory (SDRAM) or the node's main memory. To support main memory accesses, the configuration tables for the Elan MMU are synchronized with the main processor's MMU tables so that Elan can access its virtual address space. The system software is responsible for MMU table synchronization and is invisible to programmers.

The Elan MMU can translate between virtual addresses in the main processor format (for example, a 64-bit word, big-endian architecture, such as that of the AlphaServer) and virtual addresses written in the Elan format (a 32-bit word, little-endian architecture). A processor with a 32-bit architecture (for example, an Intel Pentium) requires only one-to-one mapping.

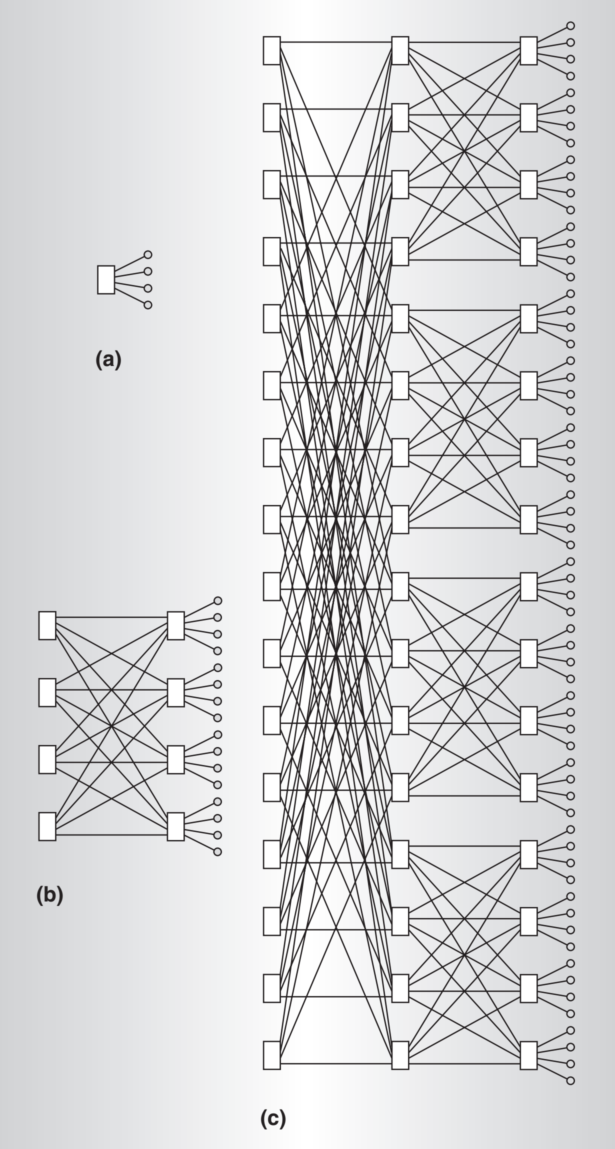

Figure 2. Quaternary *n*-trees of dimensions 1 (a), 2 (b), and 3 (c).

Figure 3 (next page) shows a 64-bit processor mapping. The 64-bit addresses starting at 0x1FF0C808000 are mapped to the Elan's 32-bit addresses starting at 0xC808000. This means that the main processor can directly access virtual addresses in the range 0x1FF0C808000 to 0x1FFFFFFFFFF, and Elan can access the same memory with addresses in the 0xC808000 to 0xFFFFFFFF

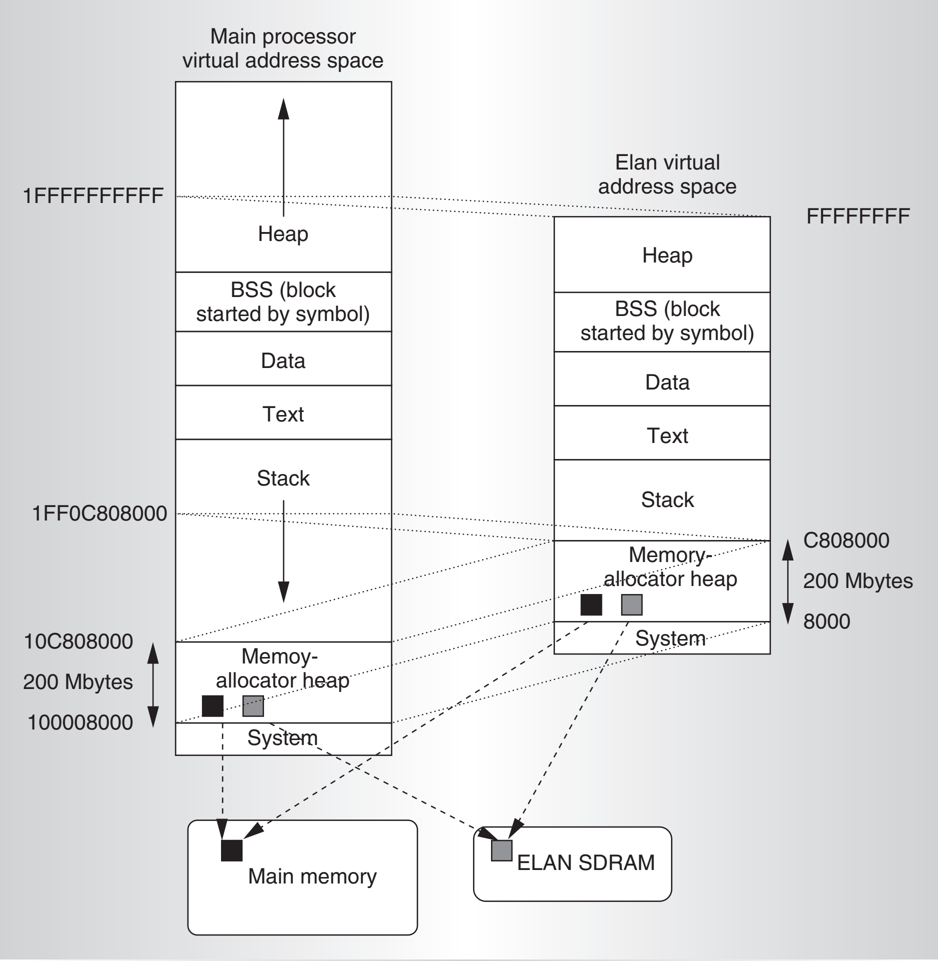

Figure 3. Virtual address translation. The dotted lines in the figure signify that a segment of memory from one address space maps onto an equally sized segment of memory in another address space.

range. In our example, the user can allocate main memory using `malloc`, and the process heap can grow outside the region directly accessible by the Elan, which is delimited by 0x1FFFFFFFFFF. To avoid this problem, both the main and Elan memory can be allocated using a consistent memory allocation mechanism.

As shown in Figure 3, the MMU tables can map a common region of virtual memory called the memory allocator heap. The allocator maps, on demand, physical pages—of either main or Elan memory—into this virtual address range. Thus, using allocation functions provided by the Elan library, the user can allocate portions of virtual memory either from main or Elan memory, and the main processor and Elan MMUs can be kept consistent.

For efficiency, programmers can locate some objects—for example, communication buffers or DMA descriptors—on the Elan memory. This way, Elan can process them independently of the main processor.

Elan context. In a conventional virtual-memory system, each user process has an assigned process identification number that selects the MMU table set and, therefore, the physical address spaces accessible to the user process. QsNet extends this concept so that the user address spaces in a parallel program can intersect. Elan replaces the process identification number value with a context value. User processes can directly access an exported segment of remote memory using a context value and a virtual address. Furthermore, the context value also determines which remote processes can access the address space via the Elan network and where those processes reside. If the user process is multithreaded, the threads will share the same context just as they share the same main-memory address space. If the node has multiple physical CPUs, then different CPUs can execute the individual threads. However, the threads will still share the same context.

Network fault detection and fault tolerance

QsNet implements network fault detection and tolerance in hardware. (It is important to note that this fault detection and tolerance occurs between two communicating Elans). Under normal operation, the source Elan transmits a packet (that is, route information for source routing followed by one or more transactions). When the receiver in the destination Elan receives a transaction with an ACK Now flag, it means that this transaction

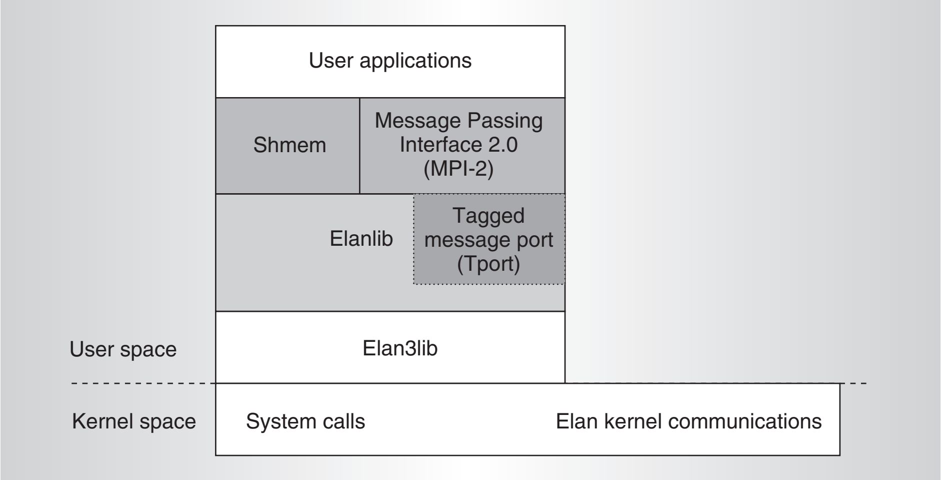
User applications
Shmem
Message Passing Interface 2.0 (MPI-2)
Elanlib
Tagged message port (Tport)
User space
Elan3lib
Kernel space
System calls
Elan kernel communications

Figure 4. Elan programming libraries.

is the last one for the packet. The destination Elan then sends a packet acknowledgment token back to the source Elan. Only when the source Elan receives the packet acknowledgment token does it send an end-of-packet token to indicate the packet transfer's completion. The fundamental rule of Elan network operation is that for every packet sent down a link, an Elan interface returns a single packet-acknowledgment token. The network will not reuse the link until the destination Elan sends such a token.

If an Elan detects an error during a packet transmission over QsNet, it immediately sends an error message without waiting for a packet-acknowledgment token. If an Elite detects an error, it automatically transmits an error message back to the source and the destination. During this process, the source and destination Elans and the Elites between them isolate the faulty link and/or switch via per-hop fault detection;[7] the source receives notification about the faulty component and can retry the packet transmission a default number of times. If this is unsuccessful, the source can appropriately reconfigure its routing tables to avoid the faulty component.

Programming libraries

Figure 4 shows the different programming libraries for the Elan network interface. These libraries trade off speed with machine independence and programmability. The Elan3lib provides the lowest-level, user space programming interface to the Elan3. At this level, processes in a parallel job can communicate through an abstraction of distributed, virtual, shared memory. Each process in a parallel job is allocated a virtual process identification (VPID) number and can map a portion of its address space into an Elan. These address spaces, taken in combination, constitute a distributed, virtual, shared memory. A combination of a VPID and a virtual address can provide an address for remote memory (that is, memory on another processing node belonging to a process). The system software and the Elan hardware use VPID to locate the Elan context when they perform a remote communication. Since Elan has its own MMU, a process can select which part of its address space should be visible across the network, determine specific access rights (for example, write or read only), and select the set of potential communication partners.

Elanlib is a higher-level interface that releases the programmer from the revision-dependent details of Elan and extends Elan3lib with point-to-point, tagged message-passing primitives (called tagged message ports or Tports). Standard communication libraries such as that of the MPI-2 standard[12] or Cray Shmem are implemented on top of Elanlib.

Elan3lib

The Elan3lib library supports a programming environment where groups of cooperating processes can transfer data directly, while protecting process groups from each other in hardware. The communication takes place at the user level, with no copy, bypassing the operating system. The main features of Elan3lib are the memory mapping and allocation scheme (described previously), event notification, and remote DMA transfers.

Events provide a general-purpose mechanism for processes to synchronize their actions. Threads running on Elan and processes running on the main processor can use this mechanism. Processes, threads, packets, and so on can access events both locally and remotely. In this way, intranetwork synchronization of processes is possible, and events can indicate the end of a communication operation, such as the completion of a remote DMA. QsNet stores events in Elan memory to guarantee atomic execution of the synchronization primitives. (The current PCI bus implementations cannot guarantee atomic execution, so it is not possible to store events in main memory.) Processes can wait for an event to by triggered by blocking, busy-waiting, or polling. In addition, processes can tag an event as block copy. The block-copy mechanism works as follows: A process can initialize a block of data in Elan memory to hold a predefined value. An equivalent-sized block is located in main memory, and both blocks are in the user's virtual address space. When the specified event is set—for example when a DMA transfer has completed—a block copy takes place. That is, the hardware in the Elan—the DMA engine—copies the block in Elan memory to the block in main memory. The user process polls the block in main memory to check its value (by, for example, bringing a copy of the corresponding memory block into the level-two cache) without polling for this information across the PCI bus. When the value is the same as that initialized in the source block, the process knows that the specified event has occurred.

The Elan supports remote DMA transfers across the network, without any copying, buffering, or operating system intervention. The process that initiates the DMA fills out a DMA descriptor, which is typically allocated on the Elan memory for efficiency. The DMA descriptor contains source and destination process VPIDs, the amount of data, source and destination addresses, two event locations (one for the source and the other for the destination process), and other information that enhances fault tolerance. Figure 5 outlines the typical steps of remote DMA. The command processor referred to in the figure is an Elan microcode thread that processes user commands; it is not a specific microprocessor.

Elanlib and Tports

Elanlib is a machine-independent library that integrates the main features of Elan3lib with Tports. Tports provide basic mechanisms for point-to-point message passing. Senders can label each message with a tag, sender identity, and message size. This information is known as the envelope. Receivers can receive their messages selectively, filtering them according to the sender's identity and/or a tag on the envelope. The Tports layer handles communication via shared memory for processes on the same node. The Tports programming interface is very similar to that of MPI.

Tports implement message sends (and receives) with two distinct function calls: a nonblocking send that posts and performs the message communication, and a blocking send that waits until the matching start send is completed, allowing implementation of different flavors of higher-level communication primitives.

Tports can deliver messages synchronously and asynchronously. They transfer synchronous messages from sender to receiver with no intermediate system buffering; the message does not leave the sender until the receiver requests it. QsNet copies asynchronous messages directly to the receiver's buffers if the receiver has requested them. If the receiver has not requested them, it copies asynchronous messages into a system buffer at the destination.

Experiments

We tested QsNet's main features on an experimental cluster with 16 dual-processor, symmetric multiprocessors (SMPs) equipped with 733-MHz Pentium IIIs. Each SMP uses a motherboard based on the Serverworks HE chipset with a 1-Gbyte SDRAM and two 64-bit, 66-MHz PCI slots (one of which is used by the Elan PCI card QM-400). The interconnection network is a quaternary fat tree of dimension two, composed of eight 8-port Elite switches integrated on the same board. We used the Linux 2.4.0-test7 operating system during this evaluation.

To expose the basic performance of QsNet, we wrote our benchmarks at the Elan3lib level. We also briefly analyzed the overhead

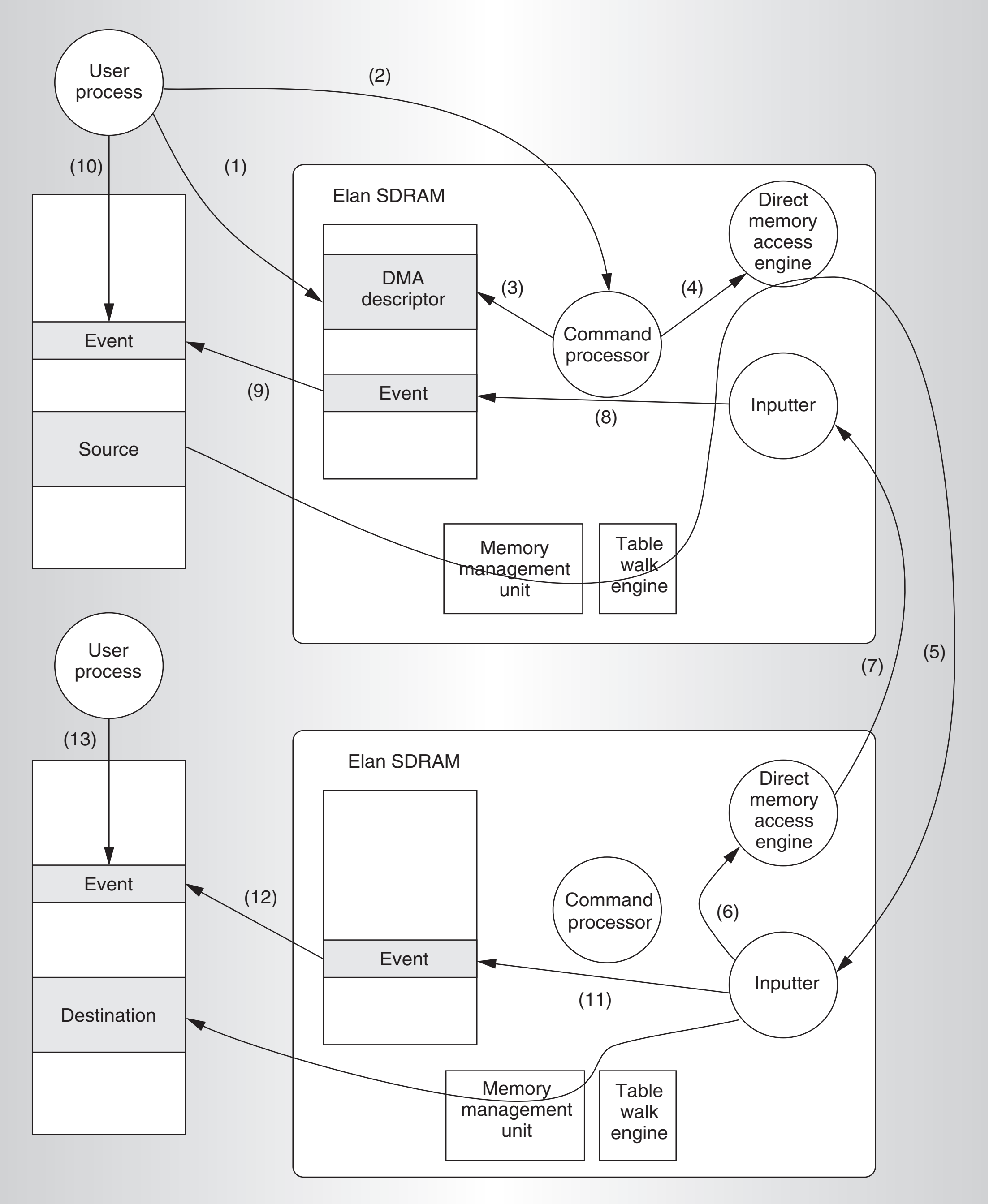

Figure 5. Execution of a remote DMA. The sending process initializes the DMA descriptor in the Elan memory (1) and communicates the address of the DMA descriptor to the command processor (2). The command processor checks the correctness of the DMA descriptor (3) and adds it to the DMA queue (4). The DMA engine performs the remote DMA transaction (5). Upon transaction completion, the remote inputter notifies the DMA engine (6), which sends an acknowledgment to the source Elan's inputter (7). The inputters or the hardware in the Elan can notify source (8, 9, 10) and destination (11, 12, 13) events, if needed.

introduced by Elanlib and an implementation of MPI-2 (based on a port of MPI-CH onto Elanlib).

To identify different bottlenecks, we placed the communication buffers for our unidirectional and bidirectional ping tests either in main

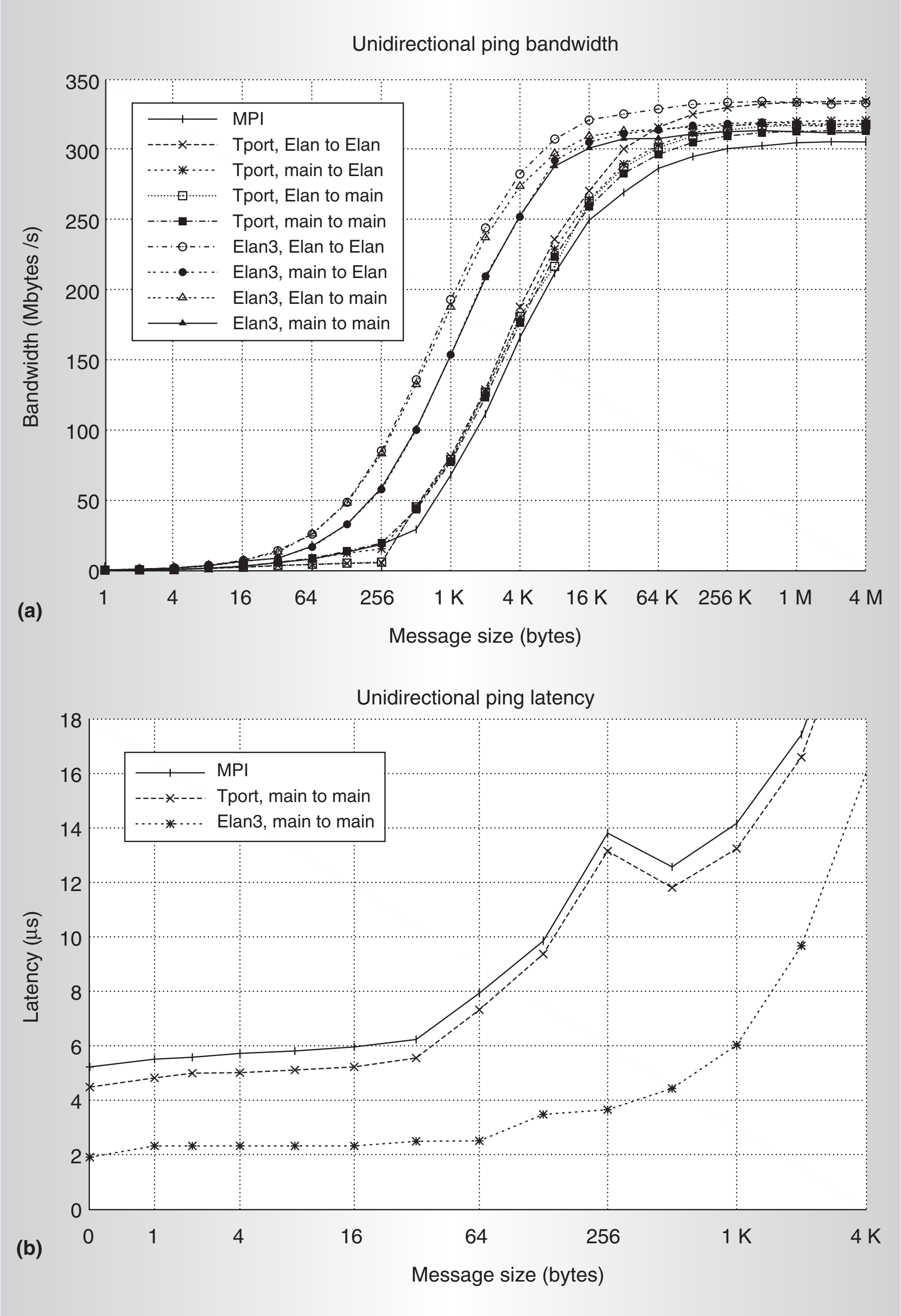

Figure 6. Unidirectional pings: bandwidth (a) and latency (b).

or Elan memory. The communication alternatives between memories include main to main, Elan to Elan, Elan to main, and main to Elan.

Unidirectional ping

Figure 6a shows the results for the unidirectional ping. The asymptotic bandwidth for all communication libraries and buffer mappings lies in a narrow range from 307 Mbytes/s for MPI to 335 Mbytes/s for Elan3lib. The results also show a small performance asymmetry between read and write performance on the PCI bus. With Elan3lib, the read and write bandwidths are 321 and 317 Mbytes/s. The system reaches a peak bandwidth of 335 Mbytes/s when we place both source and destination buffers in Elan memory.

We can logically organize the graphs in Figure 6a into three groups: those relative to Elan3lib with the source buffer in Elan memory, Elan3lib with the source buffer in main memory, and Tports and MPI. In the first group, the latency is low for small and medium-sized messages. This basic latency increases in the second group because of the extra delay to cross the source PCI bus. Finally, both Tports and MPI use the thread processor to perform tag matching, and this further increases the overhead.

Figure 6b shows the latency of messages in the range 0 to 4 Kbytes. With Elan3lib, the latency for 0-byte messages is only 1.9 µs and is almost constant at 2.4 µs for messages up to 64 bytes, because the Elan interface can pack these messages as a single write block transaction. The latency at the Tports and MPI levels increases to 4.4 and 5.0 µs. At the Elan3lib level, latency is mostly at the hardware level, whereas with Tports, system software runs as a thread in the Elan to match the message tags. This introduces the extra overhead responsible for the higher latency value. The noise at 256 bytes, shown in Figure 6b, is due to the message transmission policy. Elan inlines messages smaller than 288 bytes together with the message envelope so that they are immediately available when a receiver requests them. It always sends larger messages synchronously, and only after the receiver has posted a matching request.

Bidirectional ping

Figure 7a shows that full network bidirectionality cannot be achieved in practice. The maximum unidirectional value, obtained as half of the measured bidirectional traffic, is approximately 280 Mbytes/s, whereas, in the unidirectional case, it was 335 Mbytes/s. This gap in bandwidth exposes bottlenecks in the network and in the network interface. DMA engine interleaving with the inputter, the sharing of the Elan's internal data bus, and link-level interference in the Elite network cause this performance degradation. Counterintuitively, this 280 Mbytes/s value occurs when the source buffer is in main memory and the destination buffer is in Elan memory, rather than when both buffers are in Elan memory. In this case, the Elan memory is the bottleneck. The bidirectional bandwidth for main memory to main memory traffic is 160 Mbytes/s for all libraries. Figure 7b shows how bidirectional traffic affects latency with Elan3lib, Tports, and MPI.

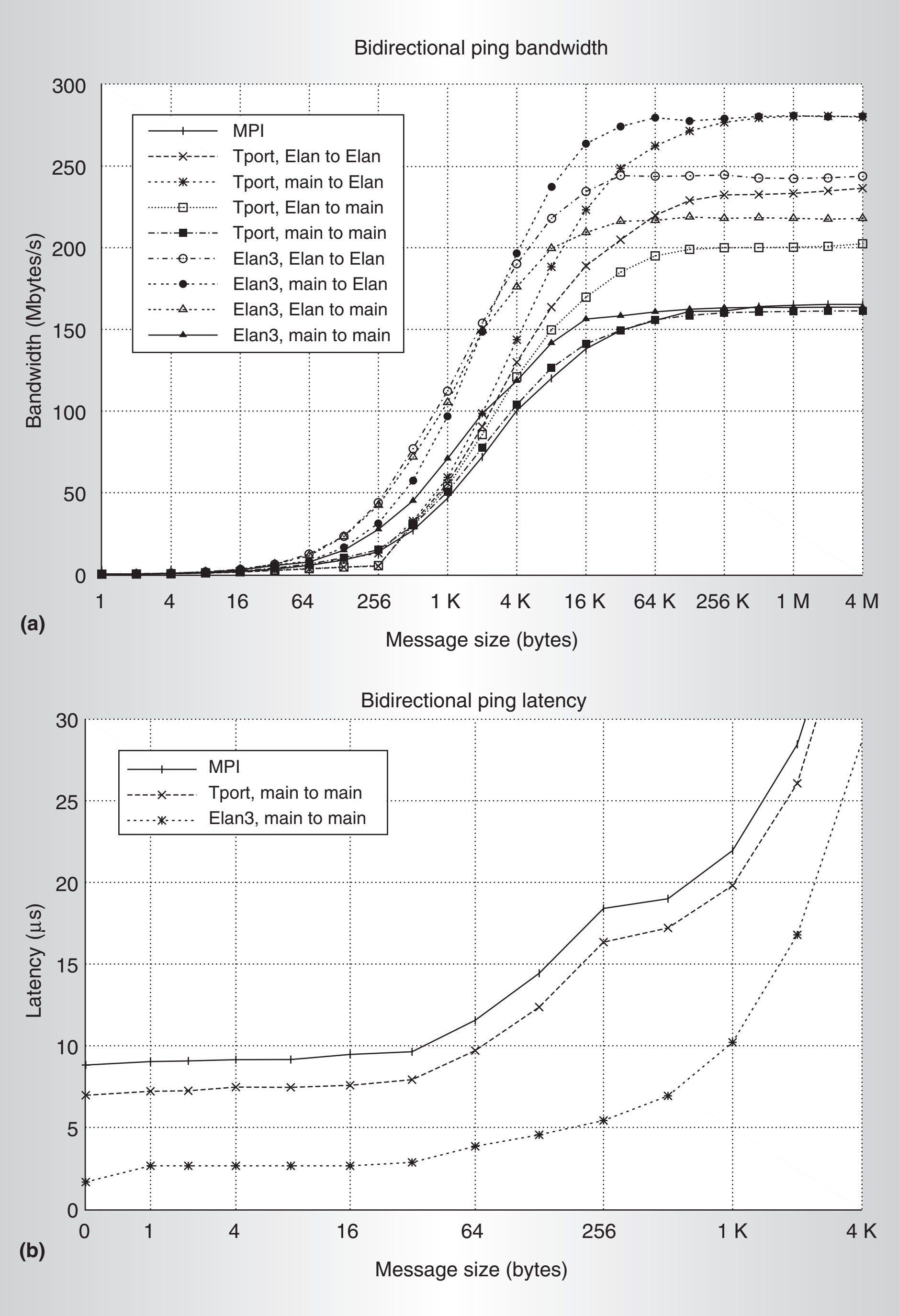

Figure 7. Bidirectional pings:bandwidth (a) and latency (b).

Hotspot

A hotspot is a single memory location that processors access repeatedly. To measure QsNet's vulnerability to such hotspots, we read from and write to the same memory location from an increasing number of processors (one per SMP). Figure 8 (next page) plots bandwidth for increasing numbers of SMPs. The upper curve of this figure shows the aggregate global bandwidth for all processes. The curves are remarkably flat, reaching 314 and 307 Mbytes/s for read and write hotspots. The lower curves show the per-SMP band-

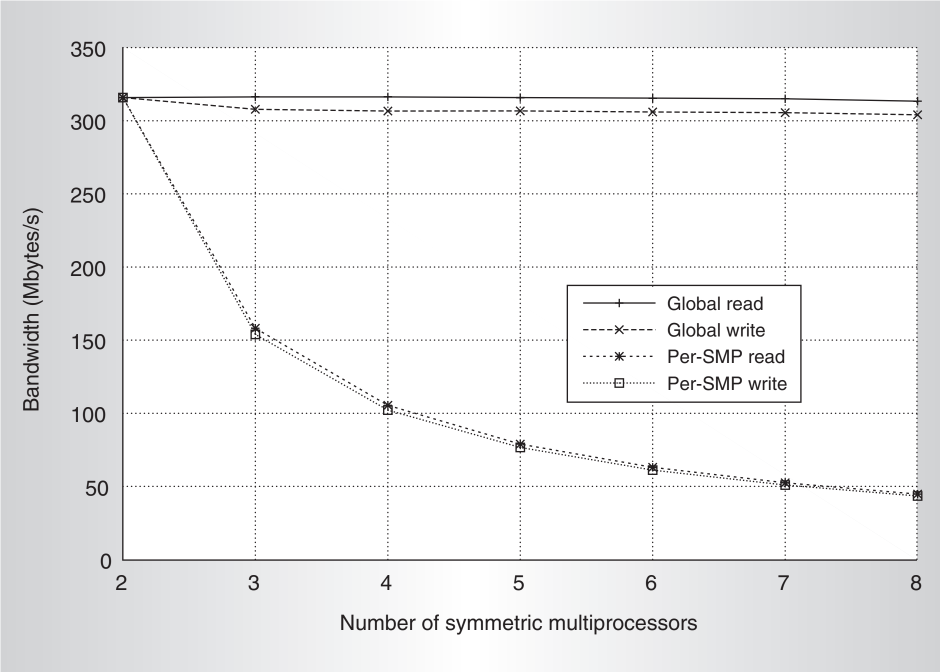

Figure 8. Bandwidth variation for read and write hotspots.

width. The scalability of this type of memory operation is very good, up to the available number of processors in our cluster.

Our analysis shows that in all components of the performance space we analyzed, the network and its libraries deliver excellent performance to users. Future work includes scalability analysis for larger configurations, performance testing of a larger subset of collective communication patterns, and performance analysis for scientific applications. MICRO

Acknowledgments

We thank the Quadrics team—David Addison, Jon Beecroft, Robin Crook, Moray McLaren, David Hewson, Duncan Roweth, and John Taylor—for their generous support. This work was supported by the US Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36.

References

1. R. Seifert, *Gigabit Ethernet: Technology and Applications for High Speed LANs*, Addison Wesley, Reading, Mass., 1998.
2. W. Vogels et al., "Tree-Saturation Control in the AC^3 Velocity Cluster," *Hot Interconnects 8*, Aug. 2000; http://www.cs.cornell.edu/vogels/clusters/ac3/hoti_files/frame.htm (current Dec. 2001).
3. H. Hellwagner, "The SCI Standard and Applications of SCI," *SCI: Scalable Coherent Interface*, *Lecture Notes in Computer Science*, vol. 1291, H. Hellwagner and A. Reinfeld, eds., Springer-Verlag, Heidelberg, Germany, 1999, pp. 95-116.
4. N.J. Boden et al., "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, Jan. 1995, pp. 29-36.
5. D. Tolmie et al., "HiPPI-800 to HiPPI-6400: A Changing of the Guard and Gateway to the Future," *Proc. 6th Int'l Conf. Parallel Interconnects* (PI), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 194-201.
6. D. Cassiday, "InfiniBand Architecture Tutorial," *Hot Chips 12*, Aug. 2000; http://www.cs.cf.ac.uk/lgds/distagen/infiniband.pdf (current Dec. 2001).
7. *Elan Programming Manual*, Quadrics Supercomputers World Ltd., Bristol, England, UK, Jan. 1999; http://www.cineca.it/manuali/QSW/docs/html/ElanProgMan/ElanProgMan.html.
8. C.E. Leiserson, "Fat-Trees: Universal Networks for Hardware Efficient Supercomputing," *IEEE Trans. Computers*, vol. C-34, no. 10, Oct. 1985, pp. 892-901.

9. F. Petrini and M. Vanneschi, "Performance Analysis of Wormhole Routed *k*-ary *n*-Trees," *Int'l J. Foundations Computer Science*, vol. 9, no. 2, June 1998, pp. 157-177.
10. W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, vol. C-36, no. 5, May 1987, pp. 547-553.
11. F. Petrini et al., "Hardware- and Software-Based Collective Communication on the Quadrics Network," *Proc. IEEE Int'l Symp. Network Computing and Applications* (NCA), IEEE Press, Piscataway, N.J., 2001, pp. 24-35.
12. W. Gropp et al., *MPI—The Complete Reference, Volume 2, The MPI Extensions*, MIT Press, Cambridge, Mass., 1998.

Fabrizio Petrini is a member of the Modeling, Algorithms, and Informatics Group (CCS-3) of the Los Alamos National Laboratory. His research interests include high-performance interconnection networks and network interfaces, job scheduling algorithms, simulation of parallel architectures, and operating systems. Petrini received a Laurea and a PhD in computer science, from the University of Pisa, Italy. He is a recipient of the European Community Marie Curie Research Fellowship. He is a member of the IEEE Computer Society.

Wu-chun Feng is a technical staff member and team leader of Research and Development in Advanced Network Technology (Radiant) at Los Alamos National Laboratory. His research interests include high-performance networking, network protocols and architecture, and cybersecurity. Feng has a BS in computer engineering and music and an MS in computer engineering from Penn State University and a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE and the ACM.

Adolfy Hoisie is a staff scientist and the Leader of the Modeling, Algorithms and Informatics Group in the Computer and Computational Sciences Division at the Los Alamos National Laboratory. His research interests include performance analysis and modeling, distributed computing, and parallel computer architectures. He is a member of the IEEE.

Salvador Coll is a faculty member in electronic engineering at the Technical University of Valencia, Spain. His research interests include multicomputer systems and interconnection networks, with a special interest in routing algorithms and collective communications. Coll has a BS in electronic engineering and an MS in computer science from the Technical University of Valencia. He is member of the Spanish Academy of Engineering.

Eitan Frachtenberg is a research assistant at Los Alamos National Laboratory and a graduate student at Hebrew University of Jerusalem, Israel. His research interests include operating systems, networks, and data compression. Frachtenberg has an MS in computer science and mathematics from the Hebrew University of Jerusalem. He is a member of the IEEE.

Direct questions and comments about this article to Fabrizio Petrini, Computer and Computational Sciences Division, Modeling, Algorithms, and Informatics (CCS-3), MS B256, Los Alamos National Laboratory, Los Alamos, NM 87545; fabrizio@lanl.gov. For additional information and related publications on the Quadrics network, please visit http://www.c3.lanl.gov/~fabrizio/quadrics.html.