The Quadrics Network (QsNet): High-Performance Clustering Technology*

Fabrizio Petrini, Wu-chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg
Computer & Computational Sciences Division
Los Alamos National Laboratory
{fabrizio,feng,hoisie,scoll,eitanf}@lanl.gov

Abstract

The Quadrics interconnection network (QsNet) contributes two novel innovations to the field of high-performance interconnects: (1) integration of the virtual-address spaces of individual nodes into a single, global, virtual-address space and (2) network fault tolerance via link-level and end-to-end protocols that can detect faults and automatically re-transmit packets. QsNet achieves these feats by extending the native operating system in the nodes with a network operating system and specialized hardware support in the network interface. As these and other important features of QsNet can be found in the InfiniBand specification, QsNet can be viewed as a precursor to InfiniBand.

In this paper, we present an initial performance evaluation of QsNet. We first describe the main hardware and software features of QsNet, followed by the results of benchmarks that we ran on our experimental, Intel-based, Linux cluster built around QsNet. Our initial analysis indicates that QsNet performs remarkably well, e.g., user-level latency under 2 μs and bandwidth over 300 MB/s.

Keywords: performance evaluation, interconnection networks, user-level communication, InfiniBand, Linux, MPI.

1 Introduction

With the increased importance of scalable system-area networks for cluster (super)computers, web-server farms, and network-attached storage, the interconnection network and its associated software libraries and hardware have become critical components in achieving high performance. Such components will greatly impact the design, architecture, and use of the aforementioned systems in the future.

Key players in high-speed interconnects include Gigabit Ethernet (GigE) [11], GigaNet [13], SCI [5], Myrinet [1], and GSN (HiPPI-6400) [12]. These interconnect solutions differ from one another with respect to their architecture, programmability, scalability, performance, and ease of integration into large-scale systems. While GigE resides at the low end of the performance spectrum, it provides a low-cost solution. GigaNet, GSN, Myrinet, and SCI add programmability and performance by providing communication processors on the network interface cards and implementing different types of user-level communication protocols.

The Quadrics network (QsNet) surpasses the above interconnects in functionality by including a novel approach to integrate the local virtual memory of a node into a globally shared, virtual-memory space; a programmable processor in the network interface that allows the implementation of intelligent communication protocols; and an integrated approach to network fault detection and fault tolerance. Consequently, QsNet already possesses many of the salient aspects of InfiniBand [2], an evolving standard that also provides an integrated approach to high-performance communication.

2 QsNet

QsNet consists of two hardware building blocks: a programmable network interface called Elan [9] and a high-bandwidth, low-latency, communication switch called Elite [10]. With respect to software, QsNet provides several layers of communication libraries that trade off between performance and ease of use. These hardware and software components combine to enable QsNet to provide the following: (1) efficient and protected access to a global virtual memory via remote DMA operations and (2) enhanced network fault tolerance via link-level and end-to-end protocols that can detect faults and automatically re-transmit packets.

2.1 Elan Network Interface

The Elan network interface connects the high-performance, multi-stage Quadrics network to a processing node containing one or more CPUs. In addition to generating and accepting packets to and from the network, the Elan provides substantial local processing power to implement high-level, message-passing protocols such as MPI. The internal functional structure of the Elan, shown in Figure 1, centers around two primary processing engines: the μcode processor and thread processor.

*This work was supported by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36.

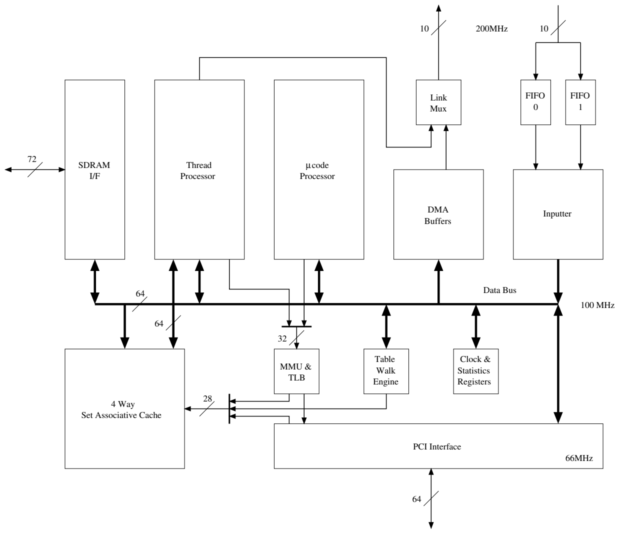

Figure 1. Elan Functional Units

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

The four μcode threads are described below: (1) *inputter thread:* Handles input transactions from the network. (2) *DMA thread:* Generates DMA packets to be written to the network, prioritizes outstanding DMAs, and time-slices large DMAs so that small DMAs are not adversely blocked. (3) *processor-scheduling thread:* Prioritizes and controls the scheduling and descheduling of the thread processor. (4) *command-processor thread:* Handles operations requested by the host (i.e., “command”) processor at user level.

The thread processor is a 32-bit RISC processor that aids in the implementation of higher-level messaging libraries without explicit intervention from the main CPU. In order to better support the implementation of high-level message-passing libraries without explicit intervention by the main CPU, its instruction set was augmented with extra instructions to construct network packets, manipulate events, efficiently schedule threads, and block save and restore a thread’s state when scheduling.

The MMU translates 32-bit virtual addresses into either 28-bit local SDRAM physical addresses or 48-bit PCI physical addresses. To translate these addresses, the MMU contains a 16-entry, fully-associative, translation lookaside buffer (TLB) and a small data-path and state machine used to perform table walks to fill the TLB and save trap information when the MMU faults.

The Elan contains routing tables that translate every virtual processor number into a sequence of tags that determine the network route. Several routing tables can be loaded in order to have different routing strategies.

The Elan has 8KB of cache memory, organized as 4 sets of 2KB, and 64MB of SDRAM memory. The cache line size is 32 bytes. The cache performs pipelined fills from SDRAM and can issue a number of cache fills and write backs for different units while still being able to service accesses for units that hit on the cache. The interface to SDRAM is 64 bits in length with 8 check bits added to provide error-code correction. The memory interface also contains a 32-byte write buffer and a 32-byte read buffer.

2.2 Elite Switch

The Elite provides (1) eight bidirectional links supporting two virtual channels in each direction, (2) an internal 16×8 full crossbar switch,[1] (3) a nominal transmission bandwidth of 400 MB/s in each link direction and a flow-through latency of 35 ns, (4) packet error detection and recovery with routing and data transactions CRC-protected, (5) two priority levels combined with an aging mechanism to ensure fair delivery of packets in the same priority level, (6) hardware support for broadcasts, and (7) adaptive routing. The switches are interconnected in a quaternary fat-tree topology, which belongs to the more general class of k-ary n-trees [7, 6].

Elite networks are source-routed, and the transmission of each packet is pipelined into the network using wormhole flow control. At the link level, each packet is partitioned in smaller units called flits (flow control digits) [3] of 16 bits. Every packet is closed by an End-Of-Packet (EOP) token, but this is normally only sent after receipt of a packet acknowledge token. This implies that every packet transmission creates a virtual circuit between source and destination.

Packets can be sent to multiple destinations using the broadcast capability of the network. For a broadcast packet to be successfully delivered a positive acknowledgment must be received from all the recipients of the broadcast group. All Elans connected to the network are capable of receiving the broadcast packet but, if desired, the broadcast set can be limited to a subset of physically contiguous Elans.

2.3 Global Virtual Memory

The Elan can transfer information directly between the address spaces of groups of cooperating processes while maintaining hardware protection between the process groups. This capability is a sophisticated extension to the conventional virtual memory mechanism and is known as *virtual operation*. Virtual operation is based on two concepts: (1) the Elan virtual memory and (2) the Elan context.

[1] The crossbar has two input ports for each input link, to accommodate two virtual channels.

2.3.1 Elan Virtual Memory

The 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 may 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 the virtual address space can be accessed by the Elan. The synchronization of the MMU tables is the responsibility of the system code and is invisible to the user programmer.

The MMU in the Elan can translate between virtual addresses written in the format of the main processor (e.g., a 64-bit word, big Endian architecture as the AlphaServer) and virtual addresses written in the Elan format (a 32-bit word, little Endian architecture). For a processor with a 32-bit architecture (e.g., an Intel Pentium), a one-to-one mapping is all that is required.

In Figure 2, the mapping for a 64-bit processor is shown. The 64-bit addresses starting at 0x1FF0C808000 are mapped to Elan's 32 bit addresses starting at 0xC808000. This means that virtual addresses in the range 0x1FF0C808000 to 0x1FFFFFFFFFF can be accessed directly by the main processor while the Elan can access the same memory by using addresses in the range 0xC808000 to 0xFFFFFFFF. In our example, the user may allocate main memory using `malloc`, and the process heap may grow outside the region directly accessible by the Elan delimited by 0x1FFFFFFFFFF. In order to avoid this problem, both main and Elan memory can be allocated using a consistent memory-allocation mechanism. As shown in Figure 2, the MMU tables can be set up to map a common region of virtual memory called the *memory-allocator heap*. The allocator maps physical pages, of either main memory or Elan into this virtual address range on demand. Thus, using allocation functions provided by the Elan library, portions of virtual memory can be allocated either from main or Elan memory, and the MMUs of both the main processor and Elan can be kept consistent.

For reasons of efficiency, some objects can be located on the Elan, e.g., communication buffers or DMA descriptors which the Elan can process independently of the main processor.

2.3.2 Elan Context

In a conventional virtual-memory system, each user process is assigned a process identification number (PID) which selects the set of MMU tables used and, therefore, the physical address spaces accessible to it. QsNet extends this concept so that the user address spaces in a parallel program can intersect. The Elan replaces the PID value with a *context* value. User processes can directly access an exported segment of remote memory by using a combination of a context value

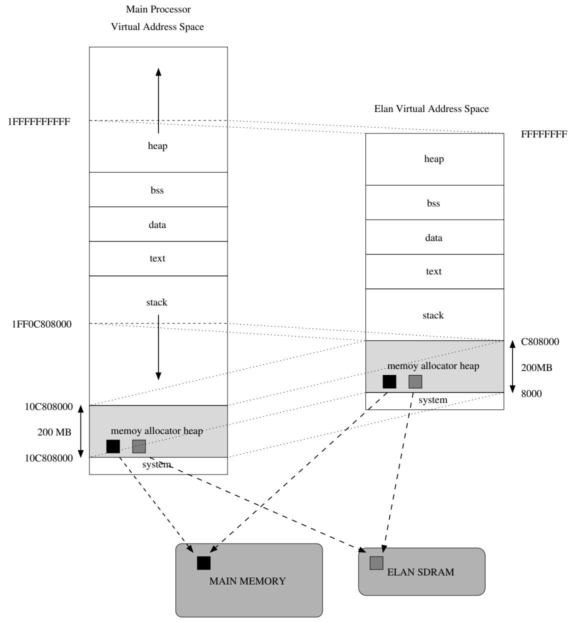

Figure 2. Virtual Address Translation

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 the individual threads may actually be executed by different CPUs. However, they will still share the same context.

2.4 Network Fault Detection & Fault Tolerance

QsNet implements network fault detection and tolerance in hardware.[2] Under normal operation, the source Elan transmits a packet (i.e., 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 it is the last transaction for the packet. The destination Elan then sends a packet acknowledgment (PA) token back to the source Elan. Only when the source Elan receives the PA token is it allowed to send an EOP acknowledgement token to the destination to indicate the completion of the packet transfer. In short, the fundamental rule of Elan network operation is that, for every packet that is sent down a link, a single PA token will be sent back. The link will not be re-used until the PA token has been sent.

If an Elan detects an error during the transmission of a packet over QsNet, it immediately sends out an error message without waiting for a PA token to be received. If an

[2] It is important to note that this fault detection and tolerance occurs between two communicating Elans.

Elite detects an error, it automatically transmits a error message back to the source and the destination. During this process, the faulty link and/or switch is isolated via per-hop fault detection [9]; the source receives notification about the faulty component and can re-try the packet transmission a default number of times. If this is not successful, the source can appropriately re-configure its routing tables so as to not use the faulty component.[3]

3 Programming Libraries

Figure 3 shows the different programming libraries [8] for the Elan network interface. These libraries trade off speed with machine independence and programmability. Starting from the bottom, Elan3lib provides the lowest-level, user-space programming interface to the Elan3. At this level, processes in a parallel job can communicate with each other through an abstraction of distributed virtual shared memory. Each process in a parallel job is allocated a virtual process id (VPID) and can map a portion of its address space into the Elan. These address spaces, taken in combination, constitute a distributed virtual shared memory. Remote memory (i.e., memory on another processing node) can be addressed by a combination of a VPID and a virtual address. Since the 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 (e.g., write- or read-only) and select the set of potential communication partners.

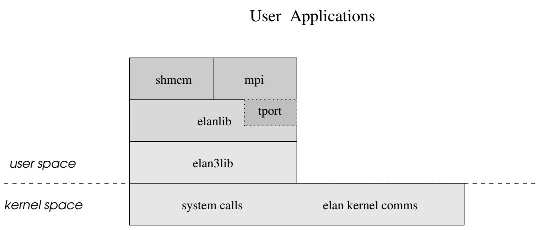

Figure 3. Elan3 Programming Libraries

Elanlib is a higher-level interface that frees the programmer from the revision-dependent details of the Elan and extends Elan3lib with point-to-point, tagged message-passing primitives (called Tagged Message Ports or Tports). Standard communication libraries as such MPI-2 or Cray `shmem` are implemented on top of Elanlib.

4 Experiments

We tested the main features of our QsNet on an experimental cluster with 16 dual-processor SMPs equipped with 733-MHz Pentium IIIs. Each SMP uses a motherboard based on the Serverworks HE chipset with 1GB of SDRAM and two 64-bit/66-MHz PCI slots (one of which is used by the Elan3 PCI card QM-400). The interconnection network is a quaternary fat-tree of dimension two, composed of eight 8-port Elite switches integrated in the same board. The operating system used during the evaluation is Linux 2.4.0-test7.

To expose the basic performance of QsNet, we wrote our benchmarks at the Elan3lib level. We also briefly analyze the overhead introduced by Elanlib and an implementation of MPI-2 [4] (based on a port of MPI-CH onto Elanlib).

To identify different bottlenecks, the communication buffers for our unidirectional ping, bidirectional ping, and hotspot tests are placed either in main or in Elan memory. The communication alternatives include main memory to main memory, Elan memory to Elan memory, Elan memory to main memory, and main memory to Elan memory.

4.1 Unidirectional Ping

Figure 4(a) shows the results for unidirectional ping. The asymptotic bandwidth for all communication libraries and buffer mappings lies in a narrow range from 307 MB/s for MPI to 335 MB/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 MB/s and 317 MB/s, respectively. The peak bandwidth of 335 MB/s is reached when both source and destination buffers are placed in the Elan memory.

The graphs in Figure 4(a) can be logically organized 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 is increased in the second group by the extra delay to start the remote DMA over the PCI bus. Finally, both Tports and MPI use the thread processor to perform tag matching, and this increases the overhead further.

Figure 4(b) shows the latency of messages in the range $[0 \ldots 4KB]$. 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 these messages can be packed as a single write-block transaction. The latency at the Tports and MPI levels increase to 4.4 and 5.0 μs, respectively. From the Elan3lib level, in which latency is mostly hardware, system software is needed to run as a thread in the Elan μprocessor in order to match the message tags; this introduces the extra overhead responsible for the higher latency value. The noise at 256 bytes, shown in Figure 4(b), is due to the message transmission policy. Messages smaller than 288 bytes are inlined together with the message envelope so that they are available immediately when a receiver posts a request for them. Larger messages are always sent synchronously, only after the receiver has posted a matching request.

[3] Recall: QsNet is source-routed.

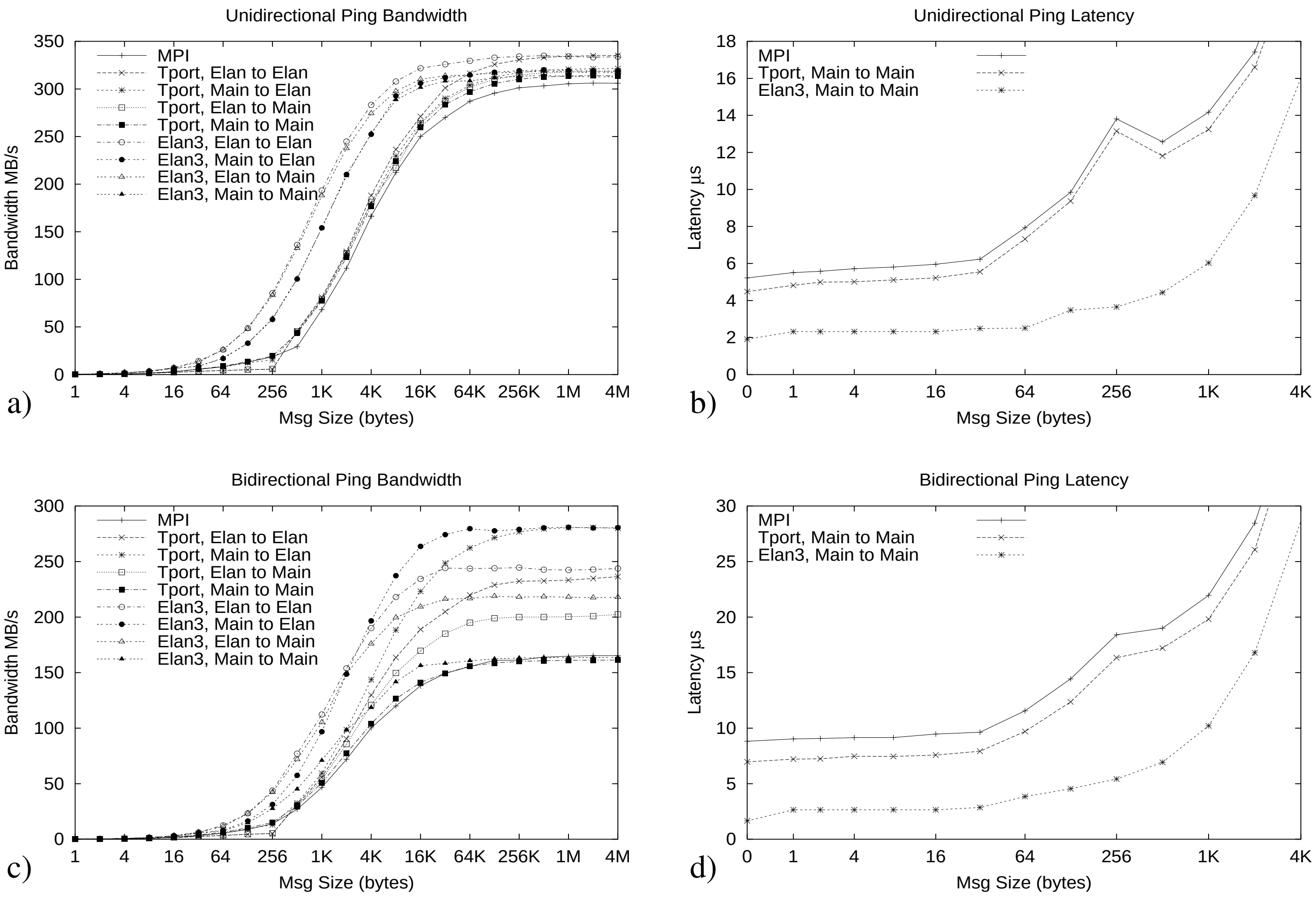

Figure 4. Unidirectional and Bidirectional Pings

4.2 Bidirectional Ping

Figure 4(c) shows that the claimed bidirectionality of the network is not fully achievable. The maximum unidirectional value, obtained as $1/2$ of the measured bidirectional traffic, is about 280 MB/s whereas in the previous case it was 335 MB/s. This gap in bandwidth exposes bottlenecks in the network and in the network interface, as opposed to the PCI bus. The causes of this performance degradation are the interleaving of the DMA engine with the inputter, the sharing of the internal data bus of the Elan, and interference at the link level in the Elite network. Counter-intuitively, this value is achieved when the source buffer is in main memory and the destination buffer in Elan memory and not when both buffers are in Elan memory. In this case, the Elan memory is the bottleneck. The bidirectional bandwidth for the main memory to main memory traffic is 160 MB/s for all libraries. Figure 4(d) shows how the bidirectional traffic affects latency with Elan3lib, Tports, and MPI.

4.3 Hotspot

In this experiment, we read from and write to the same memory location from an increasing number of processors (one per SMP). The bandwidth plots are shown in Figure 5. The upper curves are the aggregate bandwidth of all processes. The curves are remarkably flat, reaching 314 and 307 MB/s, respectively, for read and write hotspots. The lower curves show the per-SMP bandwidth. The scalability of this type of memory operation is very good, up to the available number of processors in our cluster.

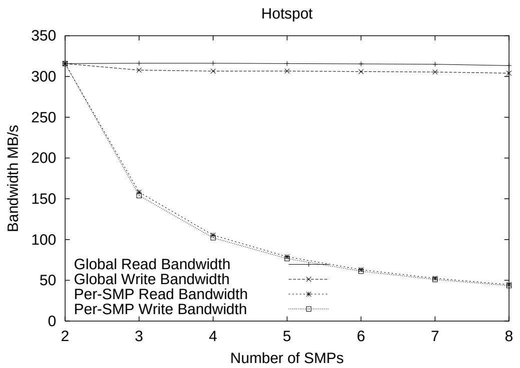

Figure 5. Read and Write Hotspot

5 Conclusion

In this paper, we presented two novel innovations of QsNet: (1) the integration of the virtual-address spaces of individual nodes into a single, global, virtual-address space and (2) network fault tolerance that can detect faults and automatically re-transmit packets. Next, we briefly presented the results of benchmark tests on QsNet, targeting essential performance characteristics. At the lowest level of the communication hierarchy, the unidirectional latency is as low as 2 μs and the bandwidth as high as 335 MB/s. Bidirectional measurements indicate a degradation in performance which we analyzed and explained in the paper. At higher levels in the communication hierarchy, Tports still exhibit excellent performance figures, comparable to the ones at Elan3lib level. In summary, our analysis shows that in all the components of the performance space we analyzed, the network and its libraries deliver excellent performance to the end user.

Future work includes scalability analysis for larger configurations, performance of a larger subset of collective communication patterns, and performance analysis of scientific applications.

References

[1] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawick, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, January 1995.

[2] Daniel Cassiday. Infiniband architecture tutorial. Hot Chips 12 Tutorial, August 2000.

[3] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[4] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI - The Complete Reference*, volume 2, The MPI Extensions. The MIT Press, 1998.

[5] Hermann Hellwagner. The SCI Standard and Applications of SCI. In Hermann Hellwagner and Alexander Reinfeld, editors, *SCI: Scalable Coherent Interface*, volume 1291 of *Lecture Notes in Computer Science*, pages 95–116. Springer-Verlag, 1999.

[6] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings of the 11th International Parallel Processing Symposium, IPPS'97*, pages 87–93, Geneva, Switzerland, April 1997.

[7] Fabrizio Petrini and Marco Vanneschi. Performance Analysis of Wormhole Routed k-ary n-trees. *International Journal on Foundations of Computer Science*, 9(2):157–177, June 1998.

[8] Quadrics Supercomputers World Ltd. *Elan Programming Manual*, January 1999.

[9] Quadrics Supercomputers World Ltd. *Elan Reference Manual*, January 1999.

[10] Quadrics Supercomputers World Ltd. *Elite Reference Manual*, November 1999.

[11] Rich Seifert. *Gigabit Ethernet: Technology and Applications for High Speed LANs*. Addison-Wesley, May 1998.

[12] D. Tolmie, T. M. Boorman, A. DuBois, D. DuBois, W. Feng, and I. Philp. From HiPPI-800 to HiPPI-6400: A Changing of the Guard and Gateway to the Future. In *Proceedings of the 6th International Conference on Parallel Interconnects (PI'99)*, October 1999.

[13] Werner Vogels, David Follett, Jenwi Hsieh, David Lifka, and David Stern. Tree-Saturation Control in the AC^3 Velocity Cluster. In *Hot Interconnects 8*, Stanford University, Palo Alto CA, August 2000.