Performance Evaluation of the Quadrics Interconnection Network*

Fabrizio Petrini⋆, Adolfy Hoisie⋆, Wu-chun Feng† and Richard Graham†

⋆ CCS-3 Modeling, Algorithms, & Informatics
† CCS-1 Advanced Computing
Computer & Computational Sciences Division
Los Alamos National Laboratory
{fabrizio,hoisie,feng,rlgraham}@lanl.gov

Abstract

We present an initial performance evaluation of the Quadrics interconnection network (QsNET). We describe the main hardware and software features of QsNET of relevance to the system designer and to the end user. Actual benchmarks are performed on an experimental Linux cluster. The preliminary analysis indicates that the interconnect performs remarkably well, e.g., user-level latency under 2 µs and bandwidth over 300 MB/s, efficient support for collective communication patterns, and excellent contention resolution under heavy traffic.

Keywords: Interconnection Networks, Performance Evaluation, User-level Communication, Operating System Bypass.

1 Introduction

System interconnection networks have become a critical component of the computing technology, and they are likely to have a great impact on the design, architecture, and use of future high-performance computers. Indeed, not only the sheer computational speed distinguishes high-performance computers from desktop systems, but the efficient integration of the computing nodes into tightly coupled multiprocessor systems. Network adapters, switches, and device driver software are increasingly becoming performance-critical components in modern supercomputers.

A cursory, and admittedly non-comprehensive, look at the state of the art in networking technology in high performance computers indicates a number of notable players. The list includes Gigabit Ethernet [14], Giganet [17], SCI [6], Myrinet [1], GSN (HIPPI 6400)[16][1], just to name a few.

These network solutions are different in terms of programmability, scalability, topology and performance. At the low end of the performance spectrum we have Gigabit Ethernet which provides a cost-effective solution for system area networks. Giganet, Myrinet and SCI add programmability and performance by using communication processors in the network interface and implementing several types of user-level, operating-system bypass communication protocols.

Infiniband [2] is an evolving standard that provides an integrated approach to high-performance communication by dealing with many aspects of the network architecture, including the elimination of bottlenecks in the I/O bus, programming interface, communication protocols, fault-tolerance, etc.

Some of these salient aspects of Infiniband already exist in the QsNET. In fact, the overall design of the QsNET provides many innovative aspects. These include a novel approach to integrate the local virtual memory into a distributed virtual shared memory, the presence of a programmable processor in the network interface that allows the implementation of intelligent communication protocols, an integrated approach to network fault-detection and fault-tolerance.

QsNET is the interconnect adopted by Compaq for its high performance servers. This interconnection strategy is of great importance to the Los Alamos National Laboratory because the 30 TeraOps ASCI[2] architecture will be centered around it.

This paper presents a performance analysis of the QsNET. Section 2 gives a comprehensive presentation of the two hardware building blocks, the Elan and the Elite. Section 3 discusses the hierarchy of communication libraries. In Section 4 we give a description of our benchmarking methodology and experimental apparatus, while Section 5 presents the experimental results and performance analysis. Some concluding remarks are given in section 6.

2 The QsNET

The QsNET is based on two building blocks, a programmable network interface called Elan [12] and a low-latency high-bandwidth communication switch called Elite

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

[1]See also http://public.lanl.gov/radiant/hippi.html

[2]http://www5.compaq.com/alphaserver/news/supercomputer_0822.html

[13]. Elites can be interconnected in a fat-tree topology [7]. The network has several layers of communication libraries which provide trade-offs between performance and ease of use. Other important features are hardware support for collective communication patterns and fault-tolerance.

2.1 Elan

The Elan[3] network interface links 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 also 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 microcode processor and the thread processor.

The 32-bit microcode processor supports four separate 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 microcode processor is extraordinarily lightweight, enabling a thread to wake up, schedule a new memory access on the result of a previous memory access, and then go back to sleep in as few as two system-clock cycles.

The four microcode 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 used to aid 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, the thread processor's instruction set was augmented with extra instructions that 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 the S-DRAM and is able to 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 the SDRAM has 64 bits and there are 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.

The link logic transmits and receives data from the network and outputs 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.

2.2 Elite

The other building block of the QsNET is the Elite switch. The Elite provides the following features: (1) 8 bidirectional links supporting two virtual channels in each direction, (2) an internal 16×8 full crossbar switch[4], (3) a nominal transmission bandwidth of 400 MB/s on 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 a fair delivery of packets in the same priority level, (6) hardware support for broadcasts, (7) and adaptive routing.

The Elite switches are interconnected in a quaternary fat-tree topology, which belongs to the more general class of the k-ary n-trees [9] [8]. A quaternary fat-tree of dimension n is composed of 4^n processing nodes and $n * 4^{n-1}$ switches interconnected as a delta network, and can be recursively build by connecting 4 quaternary fat trees of dimension $n - 1$.

Quaternary fat trees of dimension 1, 2 and 3 are shown in Figure 2.

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

The transmission of each packet is pipelined into the network using wormhole flow control. At link level, each packet is partitioned in smaller units called flits (flow control digits) [3] of 16 bits. Every packet is closed by and 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.

[3] This paper refers to the Elan3 version of the Elan. We will use Elan and Elan3 interchangeably throughout the paper.

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

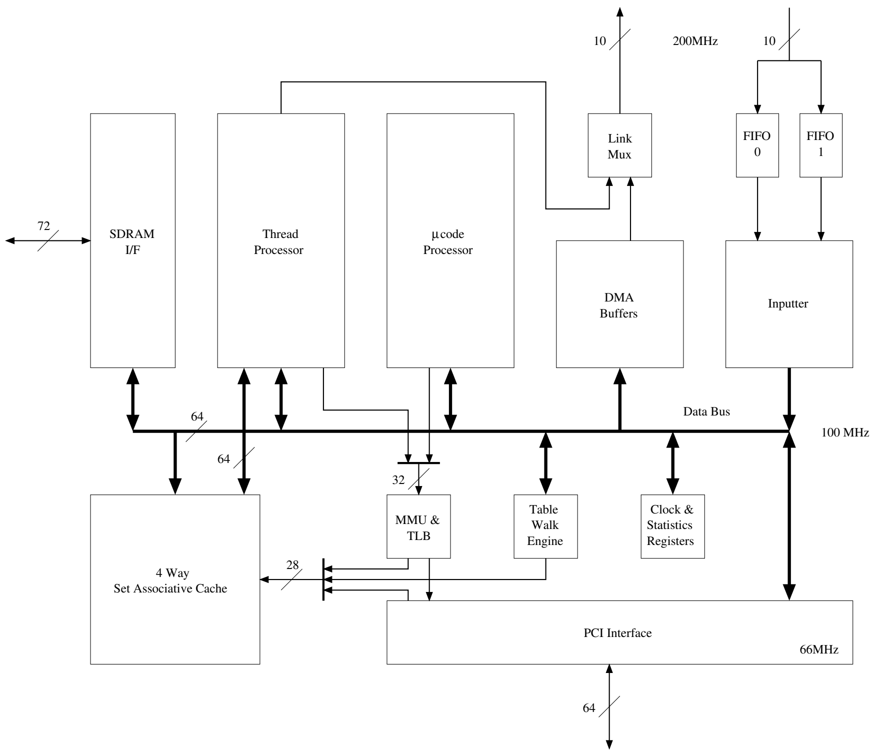

Figure 1. Elan Functional Units

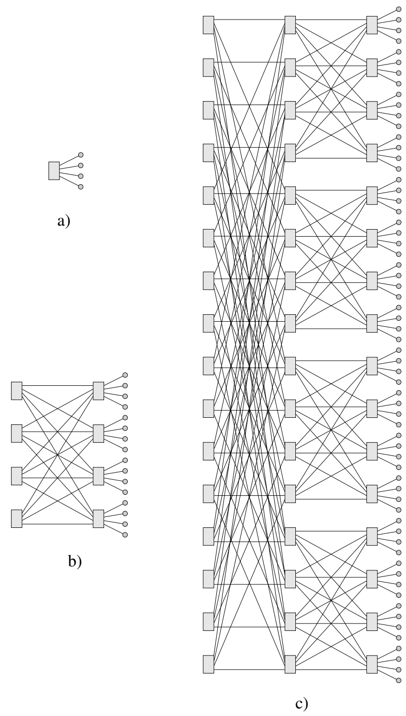

Figure 2. 4-ary n-trees of dimension 1, 2 and 3.

Packets can be sent to multiple destinations using the broadcast capability of the network. For a broadcast packet to be successfully delivered a positive acknowledgement 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.

3 Programming libraries

The Elan network interface can be programmed using several programming libraries [11], as outlined in Figure 3. These libraries trade speed with machine independence and programmability. Starting from the bottom, Elan3lib is the lowest programming level available in user space which allows the access to the low level features of 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.

Elanlib is a higher level layer 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 [4] or Cray Shmem are implemented on top of Elanlib.

3.1 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 user level, with no copy, bypassing the operating system. More information on Elan3lib is provided in Appendix A

3.2 Elanlib and Tports

Elanlib is a machine independent library that integrates the main features of Elan3lib with the Tports. Tports provide basic mechanisms for point-to-point message passing. Senders can label each message with a tag, the sender identity and the size of the message. This is known as the *envelope*. Receivers can receive their messages selectively, filtering them according to the identity of the sender and/or a tag on the envelope. The Tport layer handles communication via shared memory for processes on the same node. It is worth noting that the Tports programming interface is very similar to MPI [15].

4 Experimental Framework

We tested the main features of the QsNET on an experimental cluster with 16 dual-processor SMPs equipped with 733MHz Pentium III. Each SMP uses a motherboard based on the Serverworks HE chipset with 1GB of SDRAM. The motherboard provides two 64 bits/66Mhz PCI slots and one of them is used by the Elan3 PCI card QM-400. The interconnection network is a quaternary fat-tree of dimension two, as the one shown in Figure 2 b), with 16 external ports, the QM-S16, composed of 8 8-port Elite switches integrated in the same board. The operating system used during the evaluation is Linux 2.4.0-test7.

The preliminary results shown in this paper try to expose basic performance of the interconnection network. For this reason, most of the benchmarks are written at Elan3lib level. We will also shortly analyze the overhead introduced by Elanlib and an implementation of MPI-2 [5] which is based on a port of MPI-CH on Elanlib. A list of performed experiments follows.

4.1 Unidirectional Ping

We analyze the latency and the bandwidth of the network by sending messages of increasing size from a source to a destination SMP. In order to identify different bottlenecks, the communication buffers are placed either in main or in Elan memory. The alternatives include main memory to main memory, Elan memory to Elan memory, Elan memory to main memory and main memory to Elan memory. These buffers are placed in the desired type of memory using the allocation mechanisms provided by Elan3lib, as described in Appendix A.

For the unidirectional ping we report graphs showing latency and bandwidth. The latency is measured as the elapsed time between the posting of the remote DMA request and the notification of the successful completion at the destination (steps 2 through 5 and 11 through 13 in Figure 7). The unidirectional ping tests for the Tports and MPI are implemented using matching pairs of blocking sends and receives.

4.2 Bidirectional Ping

The unidirectional ping experiments can be considered as the "peak performance" of the network. By sending packets in both directions along the same network path we can expose several types of bottlenecks.

For example, the Elan microcode interleaves four activities, DMA engine, inputter, command processor and thread processor. This test can evaluate how the DMA engine and

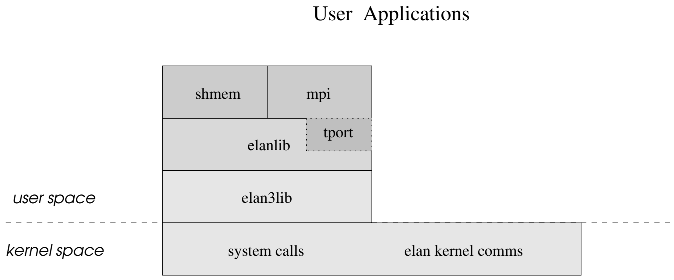

Figure 3. Elan3 programming library hierarchy

the inputter can work with bidirectional traffic. Also the link-level flow control requires the transmission of control information, which can lead to a degradation of the unidirectional performance in the presence of bidirectional traffic.

4.3 Hotspot

A further test of the network and the network interface is the hotspot. Under hotspot traffic, a set of communication partners try to read from or write into the same memory block. This localized communication pattern can lead to a severe form of congestion known as *tree saturation* [10], which can seriously degrade the performance of the overall network. In our experiments we will consider both *read* and *write* hotspots.

5 Experimental Results

5.1 Unidirectional Ping

Figure 4 a) compares the performance of the 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. The results show that there is a slight performance asymmetry between read and write performance on the PCI bus. In fact, with Elan3lib the read bandwidth is 321 MB/s while the write bandwidth is 317 MB/s. The peak bandwidth of 335 MB/s is reached when both source and destination buffers are placed in the Elan memory. The maximum amount of data payload that can be sent by the current Elan implementation is 320 bytes, partitioned in five low-level write-block transactions of 64 bytes. For this message format, the overhead is 58 bytes, for the message header, CRCs, routing info, etc. This implies that the peak bandwidth delivered by the network is approximately 396 MB/s, or 99% of the nominal bandwidth of 400 MB/s. These results also show that the PCI chipset is very efficient and is not a bottleneck for this type of communication. Other PCI chipsets, as the Intel 840[5], are significantly slower, resulting in unidirectional bandwidths of only 200 MB/s.

[5] See http://support.intel.com/support/chipsets/index.htm

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 there is a low latency with 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 furtherly increases the overhead.

Figure 4 b) shows the latency in the range $[0 \ldots 4KB]$. With Elan3lib the basic latency for 0-byte messages is only $1.9\ \mu$s and is almost constant at $2.4\ \mu$s for messages up to 64 bytes, because these messages can be packed as a single write-block transaction. We note an increase in the latency at the Tports and MPI level, compared to the latency at the Elan3lib level, from approximately $2\ \mu$s to 4.4 and $5\ \mu$s, respectively. From the Elan3lib level, in which latency is mostly hardware, system software is needed to run as a thread in the Elan microprocessor 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 the message transmission policy. In fact, messages smaller than 288 bytes are sent inline 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.

5.2 Bidirectional Ping

In this benchmark, whose results are shown in Figure 4 c), we continue to draw the curves corresponding to the experiments described in the previous case. We see that the claimed bidirectionality of the network is not fully achievable. The maximum undirectional 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 identifies 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 also interferences at link level in the Elite network. Counterintuitively, 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.

5.3 Hotspot

In this experiment we attempt to read from and write into the same memory location from an increasing number of processors (one per SMP). The bandwidth plots are depicted in Figure 5. The upper curves are the aggregate bandwidth of all processes. The curves are remarkably flat, reaching 314 MB/s and 307, respectively for read and write hotspots. The lower curves show the per-process bandwidth. The scalability of this type of memory operation is very good, up to the available number of processors in our cluster. Hotspot operations, both on read and on write, are common in scientific computing, hence the scalability of hotspot resolution is very important.

6 Conclusion and Future Work

We presented the results of several performance tests on the 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 307 MB/s. These performance numbers are influenced by the mapping of the buffers in various levels of the memory hierarchy. The network bandwidth, measured by placing the communication buffers in the Elan memory is about 335 MB/s. Bidirectional measurements indicate a degradation in performance which is 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 delivers adequate performance levels 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 ASCI 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] Al Geist, William Gropp, Steve Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, William Saphir, Tony Skjellum, and Marc Snir. MPI-2: Extending the Message Passing Interface. In *Second International Euro-Par Conference, Volume I*, number 1123 in LNCS, pages 128–135, Lyon, France, August 1996.

[5] 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.

[6] 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.

[7] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[8] 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.

[9] 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.

[10] G. F. Pfister and V. A. Norton. Hot-spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, C-34(10):943–948, October 1985.

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

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

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

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

[15] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI - The Complete Reference*, volume 1, The MPI Core. The MIT Press, 1998.

[16] 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.

[17] 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.

A Elan3lib

The main features of Elan3lib are: (1) event notification, (2) the memory mapping and allocation scheme and (3) remote DMA transfers.

A.0.1 Event Notification

Events provide a general purpose mechanism for processes to synchronize their actions. The mechanism can be used by threads running on the Elan and processes running on the

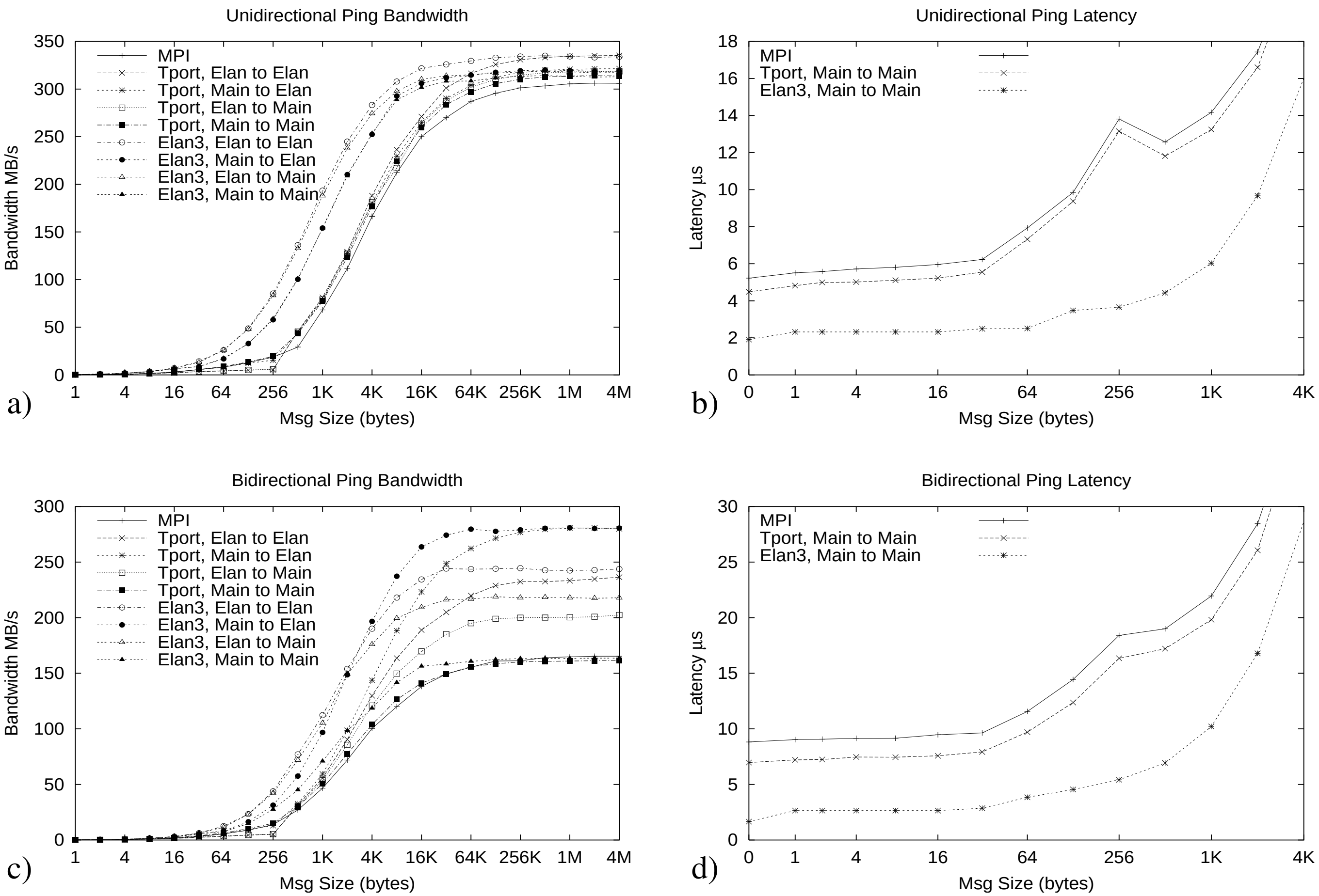

Figure 4. Unidirectional and Bidirectional Pings

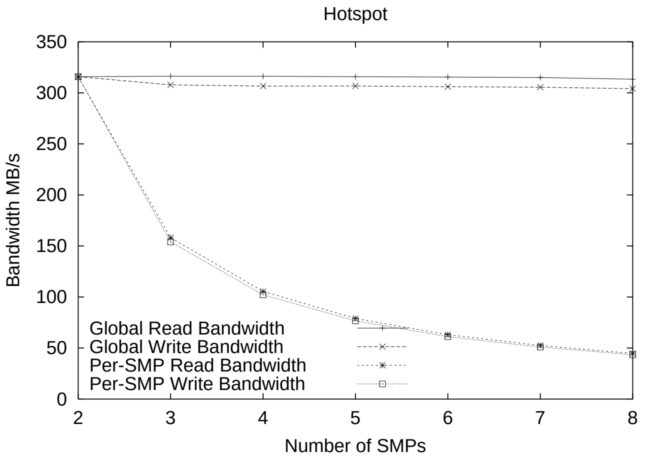

Figure 5. Read and Write Hotspot

main processor. Events can be accessed both locally and remotely. In this way, processes can be synchronized across the network, and can be used to indicate the end of a communication operation, such as a completion of a remote DMA. Events are stored in Elan memory, in order to guarantee the atomic execution of the synchronization primitives[6]. Processes can wait for an event to be triggered by blocking, busy waiting or polling. In addition, an event can be tagged as being a block copy event. The block copy mechanism works as follows. A block of data in Elan memory is initialized to hold a pre-defined value. An equivalent sized block is located in main memory, and both 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 block in Elan memory is copied to the block in main memory. The user process polls the block in main memory to check its value, (for example, bringing a copy of the corresponding memory block into the L2 cache) without having to poll 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 happened.

A.0.2 Memory Mapping and Allocation

The MMU in the Elan can translate between virtual addresses written in the format of the main processor (for example, 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 (for example an Intel Pentium), a one-to-one mapping is all that is required.

In Figure 6 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 6 the MMU tables can be set up to map a common region of virtual memory called *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 (1) can be allocated either from main or Elan memory, and (2) the MMUs of both main processor and Elan can be kept consistent.

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

A.0.3 Remote DMA

The Elan supports remote DMA (Direct Memory Access) transfers across the network, without any copying or 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 reasons. The DMA descriptor contains the VPIDs of both source and destination, the amount of data, the source and destination addresses, two event locations (one for the source and the other for the destination process) and other information used to enhance fault tolerance.

The typical steps of remote DMA are outlined in Figure 7.

[6]The current PCI bus implementations cannot guarantee atomic execution, so it is not possible to store events in main memory.

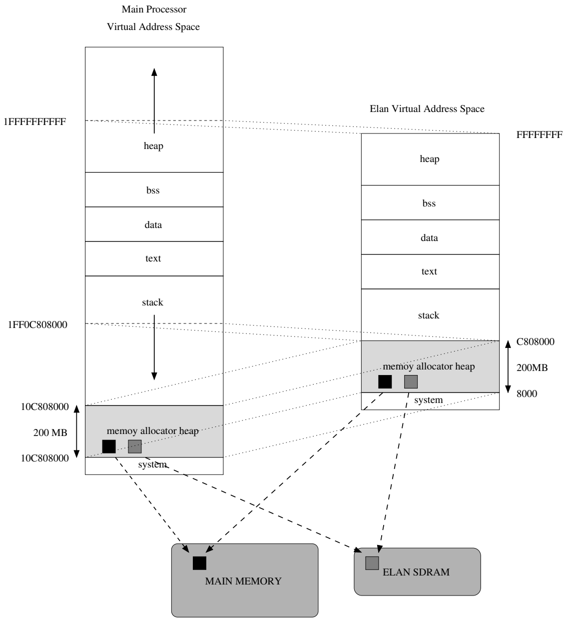

Figure 6. Virtual Address Translation

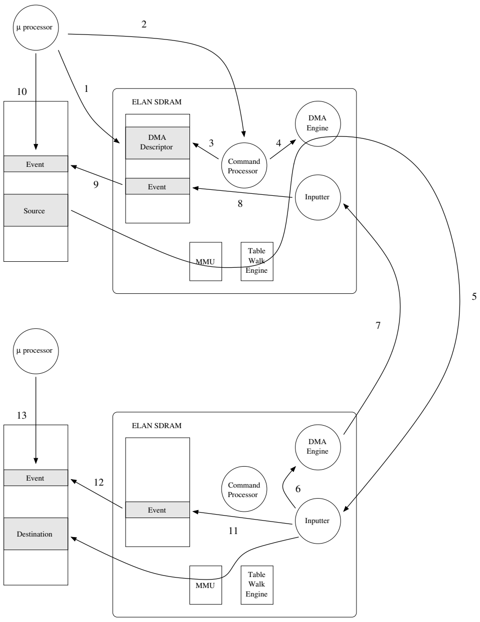

Figure 7. Execution of a Remote DMA. The sending process (1) initializes the DMA descriptor in the Elan memory and (2) communicates the address of the DMA descriptor to the command processor. The command processor (3) checks the correctness of the DMA descriptor and (4) adds it to the DMA queue. The DMA engine (5) performs the remote DMA transaction. Upon completion the remote inputter (6) notifies the DMA engine which (7) sends an ack to the source Elan. Source (8-10) and destination (11-13) events can be notified, if needed.