Los Alamos Unclassified Report: LA-UR 00-5205

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. A benchmark methodology is proposed in which important performance and scalability parameters are included. Actual benchmarks are performed on an experimental cluster. Based on these benchmarks, a thorough performance analysis of the QsNET is presented in the paper. The preliminary analysis indicates a remarkably good performance of the interconnect, such as user-level latency of 2.1 μs, a communication bandwidth of 200 MB/s with $N^{1/2}$ of 512 bytes, efficient support to collective communication patterns and good 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.

Given the tremendous increase in the size of parallel computers, network-based design of these architectures is now common place. With the increasing size of these machines the computations performed on them scaled up dramatically, leading to vast communication requirements. These applications now use and generate data sizes that stress the performance limits of the network. The network requirements of these applications are high in terms of latency, bandwidth, collective communication performance, I/O contention resolution, among others [7]. It is now common that many parallel applications of interest spend more time in communication rather than in computation [12]. This is not only due to the larger data movement requirements but also a direct result of the explosive increase in the processing speed, which is not matched by a similar increase in the network performance. The end result is that the network is becoming the bottleneck in many applications.

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 [23], Giganet [28], SCI [10], Myrinet [1], GSN (Hippi 6400)[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 [3] is an evolving standard that provides and integrated approach to high performance communication that deals 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.

[1]http://www.lanl.gov/lanp

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 [20] and a low-latency high-bandwidth communication switch called Elite [21]. Elites can be interconnected in a fat-tree topology [15]. 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] communications processor 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

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

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

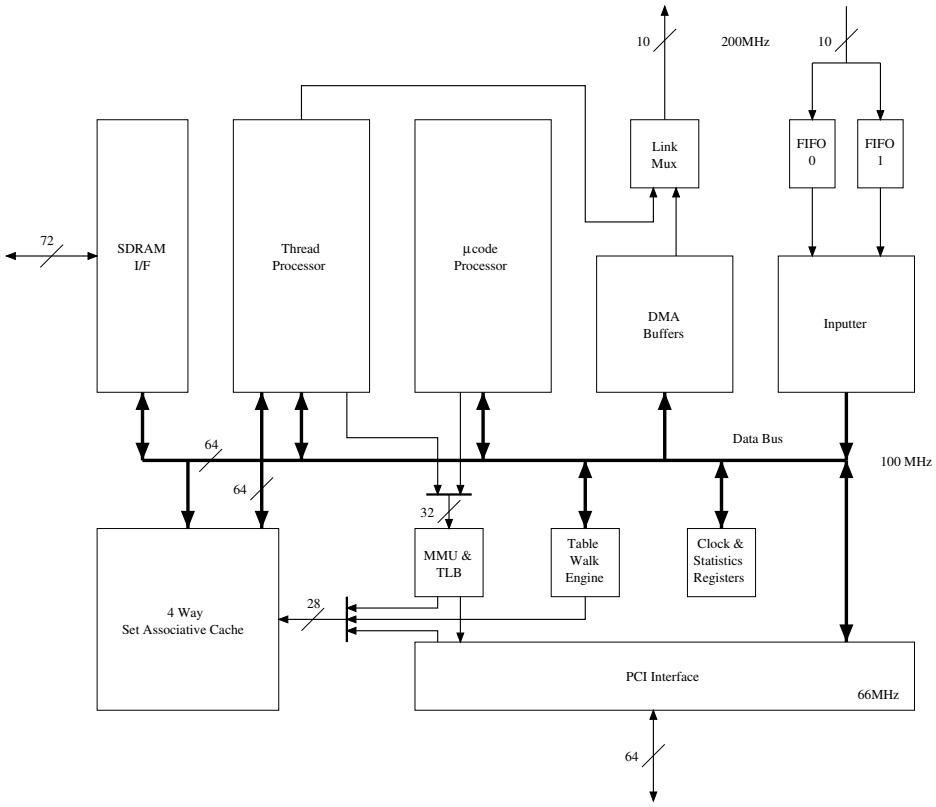

Figure 1: Elan Functional Units

the Elan, shown in Figure 1, centers around two primary processing engines: the microcode processor and the thread processor.

2.1.1 Microcode 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.

2.1.2 Thread Processor

The thread processor is a 32-bit SPARC-based RISC processor used to aid the implementation of higher-level messaging libraries without explicit intervention from the main CPU. This processor has a 32-deep register file, ALU, and shifter. Its four-stage execution pipeline consists of (1) instruction fetch, (2) instruction decode and register operand fetch, (3) instruction execute, and (4) register write-back or memory operation. 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.

2.1.3 Memory Management Unit (MMU)

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.

2.1.4 Route Fetch Engine

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.

2.1.5 Elan and Cache Memory

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 SDRAM 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.

2.1.6 Link

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:

- 8 bidirectional links supporting two virtual channels in each direction,

- an internal 16×8 full crossbar switch[4],
- a nominal transmission bandwidth of 400 MB/s on each link direction and a flow through latency of 35 ns,
- packet error detection and recovery, with routing and data transactions CRC protected,
- two priority levels combined with an aging mechanism to ensure a fair delivery of packets in the same priority level,
- hardware support for broadcasts,
- and adaptive routing.

2.2.1 Fat-Tree topology

The Elite switches are interconnected in a quaternary fat-tree topology, which belongs to the more general class of the k-ary n-trees [17] [16]. 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.

A formal description of the k-ary n-trees and their properties is provided in Appendix A.

2.2.2 Packet Routing

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.

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

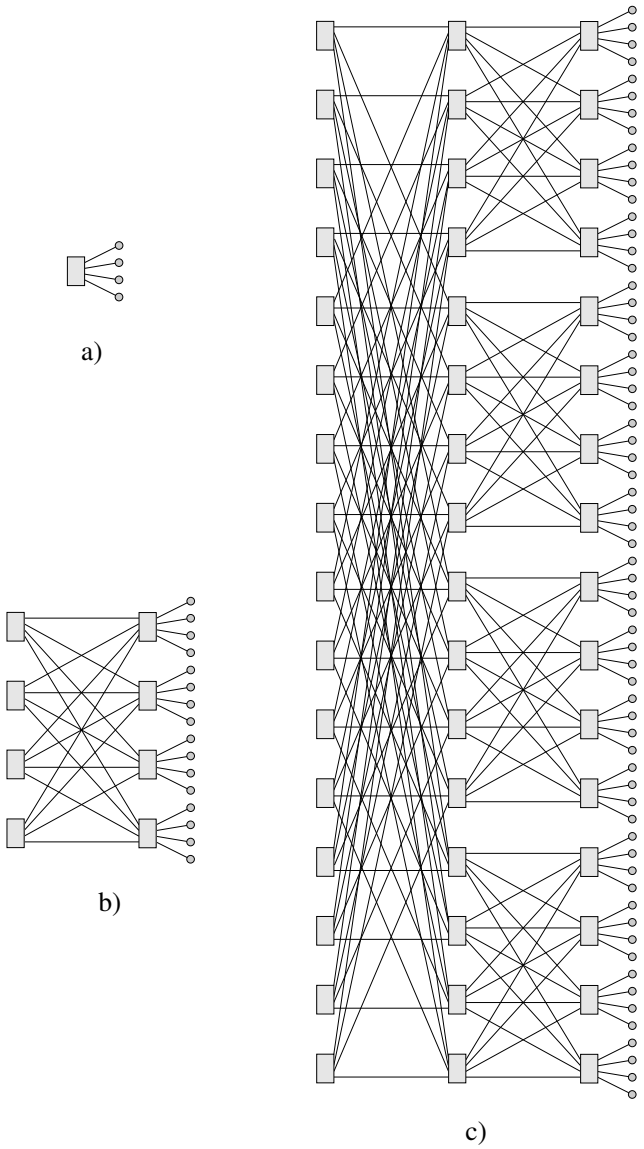

Figure 2: 4-ary n-trees of dimension 1, 2 and 3.

2.2.3 Flow Control

The transmission of each packet is pipelined into the network using wormhole flow control. At link level, each packet is partitioned in flits (flow control digits) [5] 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 create a virtual circuit between source and destination.

2.2.4 Broadcasting

Packets can be sent to multiple destinations using the broadcast capability of the Elan 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.

2.2.5 The JTAG Control Network

The Elite implements a standard JTAG test access port through which various internal registers and logic signals can read and written. Through the JTAG port it is possible to inspect the status of each single link, reset the link in case of errors, and configure various features of the Elite.

3 Programming libraries

The Elan network interface can be programmed using several programming libraries [19], 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-II [6] or Cray Shmem are implemented on top of Elanlib.

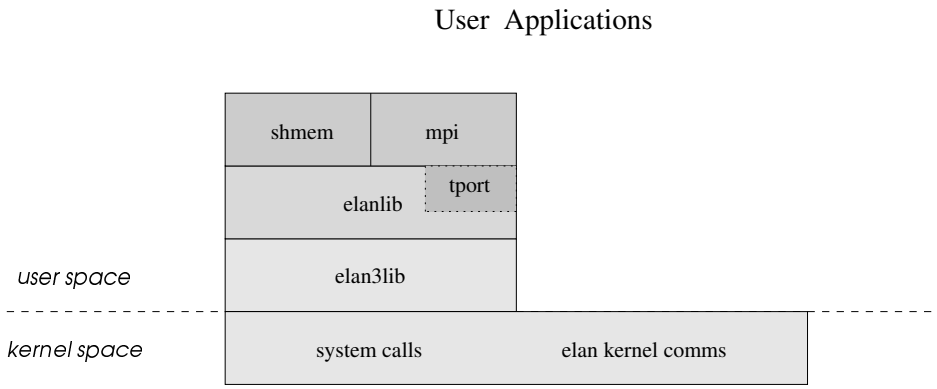

Figure 3: Elan3 programming library hierarchy

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.

The main features of Elan3lib are:

- event notification,
- the memory mapping and allocation scheme and

- remote DMA transfers.

3.1.1 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 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[5]. 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.

3.1.2 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 4 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

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

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 4 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.

3.1.3 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 5.

The process that initiates the DMA (1) fills out a DMA descriptor and (2) writes the address of the DMA descriptor in a special memory location, called command port. The command port is mapped into both user's and Elan address space, so no system calls are required to access it.

As soon as the descriptor has been written to the command port, the initiating process can continue, relying on the event mechanism to notify it when the transfer has completed. In the meantime, the Elan (3) performs all the checking required to implement interprocess protection and, if successful, (4) reads in the DMA descriptor and adds it to the DMA queue. When the

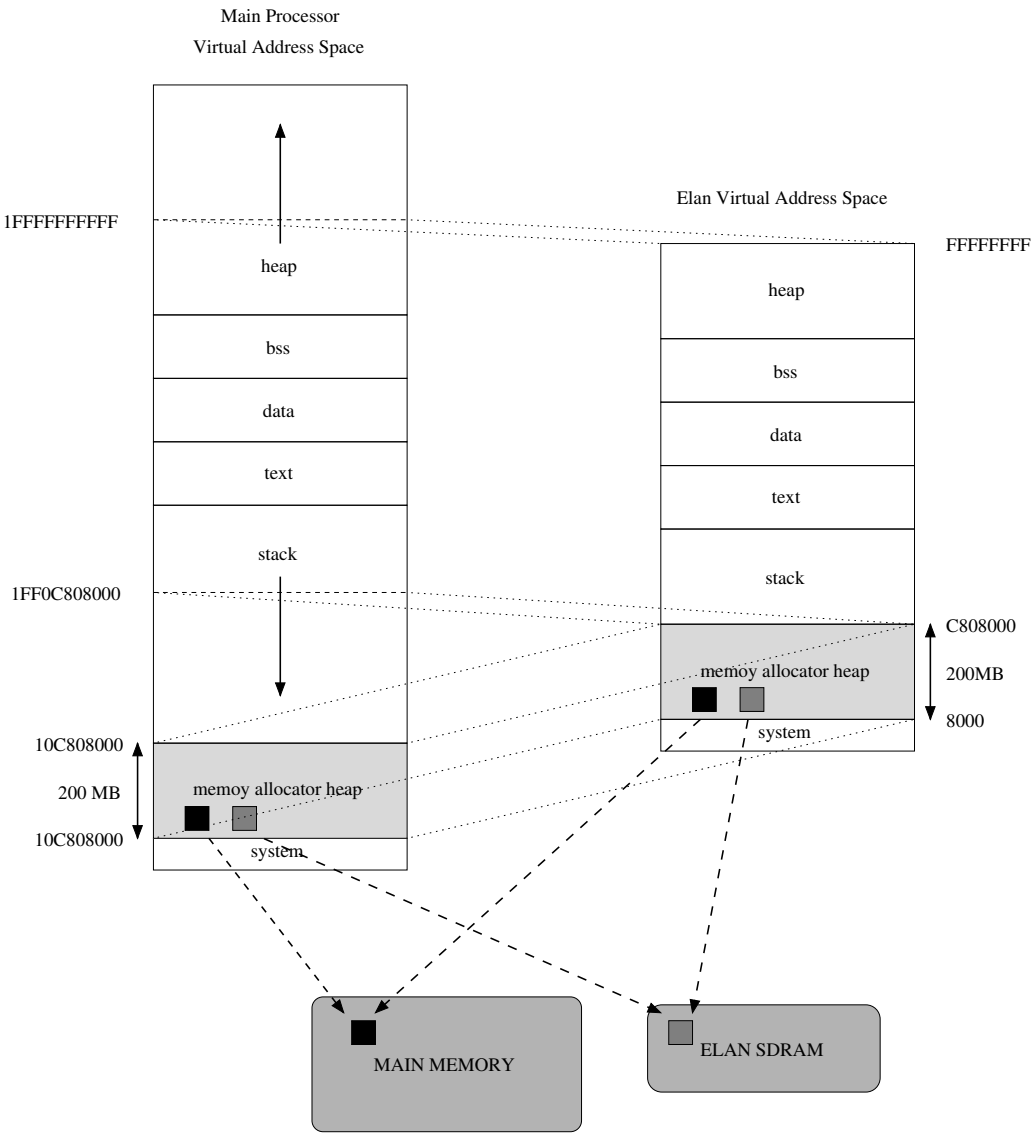

Figure 4: Virtual Address Translation

descriptor reaches the head of the line, (5) the DMA processor checks that the source and destination processes are allowed to communicate and then reads in the source data, splits it into packets, adds routing information, and sends it to the destination.

The routing info is stripped off as the packet traverses the network and the packets are reassembled by the inputter at the destination. When the transfer is complete, (6) the inputter notifies the DMA engine of the successful completion of the transaction and (7) an acknowledgement is sent back to the source Elan which then (8) sets the source event. At this point, (9) the Elan updates the copy block event in main memory and the main processor (10) can be notified of the event (for example, by polling the event location in main memory). If needed, an event (steps 11-13) can also be set at the remote destination. It is worth noting that, given the virtual addressing scheme in place in both source and destination Elans, both DMA engine and inputter can perform true zero-copy communication directly, reading from the source address/incoming virtual channel and writing into the outgoing virtual channel/destination address.

3.2 Elanlib and Tports

Elanlib is a machine independent library that integrates the main features of Elan3lib with the Tagged Message Ports or Tports. Tagged message ports 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 know 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. It is worth noting that the Tports programming interface is very similar to MPI [25].

Message sends (and receives) are implemented with two distinct function calls, a non-blocking start send, which posts and performs the message communication and a blocking call that waits until the matching start send has been completed, thus allowing the implementation of different flavors of higher level communication primitives.

Messages can be delivered synchronously and asynchronously. Synchronous messages are transferred from sender to receiver with no intermediate system buffering. The message does not leave the sender until the receiver has posted a request for it. Asynchronous messages are copied directly to the

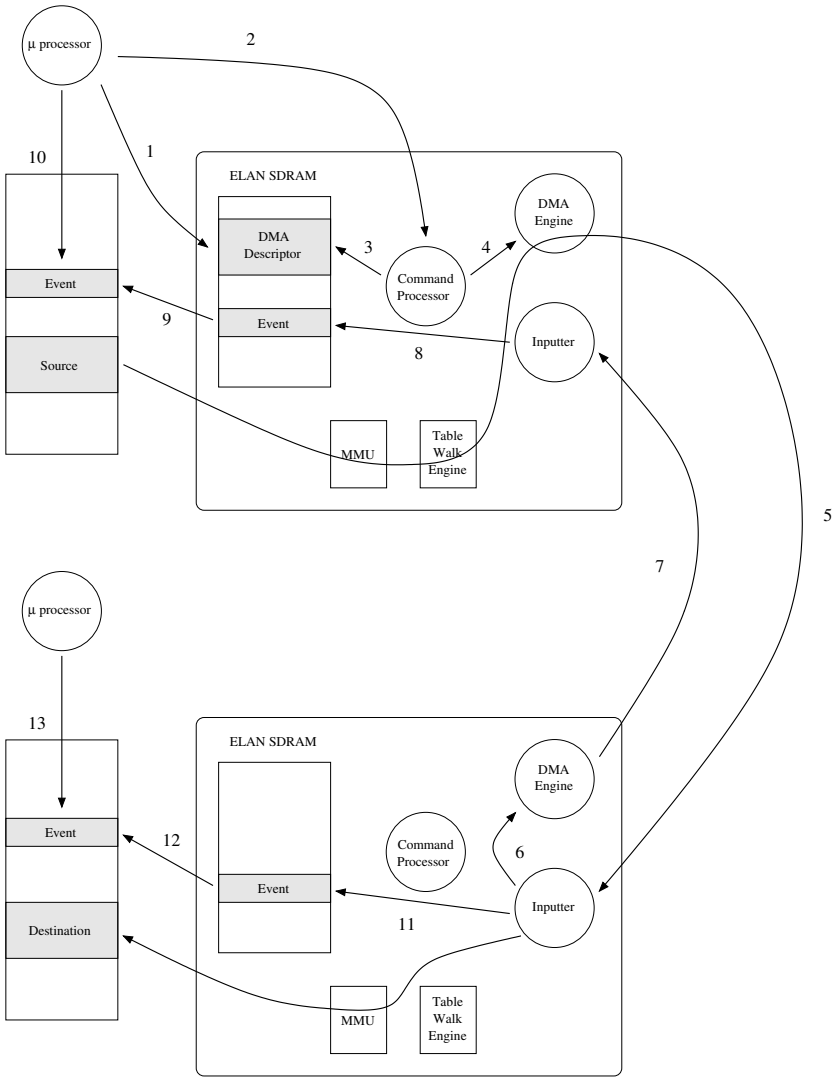

Figure 5: Execution of a Remote DMA. The sending process (1) initializes the DMA descriptor in the Elan memory and (2) communicates the address of the DMA descriptor to the command processor. The command processor (3) checks the correctness of the DMA descriptor and (4) adds it to the DMA queue. The 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.

receiver's buffers if the receiver has posted a request for them. If the receiver has not posted a request for them, asynchronous messages are copied into a system buffer at destination.

In our experiments we will evaluate an implementation of MPI-II [8] which is based on a port of MPI-CH on Elanlib and Tports.

4 Experimental Framework

We tested the main features of the QsNET on an experimental cluster with 8 dual-processor SMPs equipped with 733MHz Pentium III. Each SMP uses a motherboard based on the 840 Intel chipset with 512MB 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 and connected to the cluster host through the JTAG control port. The operating system used during the evaluation is Linux 2.3.99.

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 MPI. A list of performed experiments follows.

4.1 Latency of the Elite Switch

The hardware latency of the Elite switch is only a few tens of nanoseconds. In order to experimentally evaluate a communication latency with such small granularity, we perform the following test. We first send a 0-byte packet from a source node to a destination directly connected to the same Elite switch. We then send the same packet from the same source to a destination which can only be reached passing through one the four nearest common ancestors, traversing three distinct switches. The difference between the two values gives us a rough estimate of the latency of two switches. In order to eliminate the noise in the measurement and to avoid problems with the timer resolution (which is in the hundreds of nanoseconds) we measure the latency of a sample of one million packets and we analyze different pairs of source destination processes.

4.2 Barrier Synchronization Overhead

One of the main advantages of the fat tree topology is the possibility of implementing at hardware level fast and scalable collective communication patterns. An example is the barrier synchronization that, when executed by a group of processors that are physically adjacent in the Elite network, can use a hardware broadcast facility. In the first phase of the barrier synchronization, each participating processor enters the barrier by sending a message to a designated host (the processor with lowest physical id). When the host has received a message from all the processes involved, it can notify the end of the barrier by using the hardware broadcast.

4.3 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 Section 3.1.

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 notification of the successful completion at the source (steps 2 though 10 in Figure 5). As a consequence, the results shown in the experimental evaluation are slightly conservative for small messages. The unidirectional ping tests for the Tports and MPI are implemented using matching pairs of blocking sends and receives.

4.4 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 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.

4.5 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 a know as *tree saturation* [18], which can seriously degrade the performance of the overall network. In our experiments we will consider both *read* and *write* hotspots.

4.6 Total Exchange

The all-to-all personalized communication, or simply *total exchange*, is an important communication pattern that is at the heart of many applications, such as matrix transposition and the fast Fourier transform (FFT), and programming models such as the BSP [24]. The total exchange is a collective communication pattern where every node has to send a distinct message of the same size to all other nodes. The efficient implementation of the total-exchange has been extensively studied for a variety of networks [13] [2] [11] [26] [27] [22]. The total exchange stresses the bisection bandwidth of the network and exposes the network behavior under heavy load. A lower bound for the execution time T of the total exchange is determined by the equation

$$T = \frac{S * (N-1)}{B} \tag{1}$$

where S is message size, N the number of processors involved in the total exchange and B is the communication bandwidth of the network interface. In fact, every processor must receive $N-1$ messages of size S, and they must flow through a communication link whose bandwidth is at most B.

4.7 Description of the measurement methodology

Measurements of unidirectional bandwidth and latencies were done by timing the round-trip time for the particular event. Round-trip measurements are preferable because they exclude the need for clock synchronization. At the Elan3lib level, the round-trip consists of a put paired with an event notification (wait). The event ack consumes some bandwidth. However, due to the fact that the ack (wait) is partly overlapped with the put, and the fact that the payload of this memory access is very small, its effect on the measured bandwidth is small. In experiments involving memory accesses between different processing nodes we map the communication buffers either in the main memory or in the Elan's memory in order to bypass the PCI bus.

For Tports measurements, the round-trip for the messaging is "real", in that there is no need for an explicit ack instruction. Similarly at the MPI level. As mentioned before, at MPI level there is no possibility of targeting memory levels other than the memory in the main processor.

Bidirectional bandwidth and latency measurements involve round-trip measurements originating in each of the processor in a pair. All the other details stay the same.

Hotspot experiments for reads and writes involve many processes attempting to read or write the same memory location at once.

In all experiments, but the internal SMP pings, we use a single processor in each SMP.

5 Experimental Results

5.1 Latency of the Elite Switch

Figure 6 depicts the network latency when one and three hops (switches) in distance are involved. This gives an indication of the pure hardware latency of the switches and the wires. From the jump we infer that each hop adds approximately 50 ns in latency in one direction. Based on the technical specs [21], about 35 ns are spent in the switch and the difference is wire and protocol latency.

In a 5-dimensional 4-tree, connecting 1024 SMPs, a packet crosses at most 9 switches, which implies a worst case network latency, in the absence of contention, of about 0.5 μs. Considering that the latency in the network

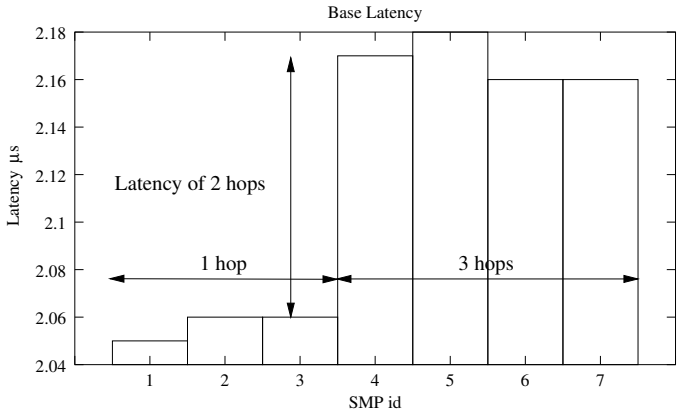

Figure 6: Network Latency

interface is at least 1 μs and that the wormhole message transmission is pipelined, we can claim that message transmission is almost insensitive to the physical location in the network of source and destination processors. Overall, this is a remarkably small latency and a remarkable network design.

5.2 Barrier Synchronization Overhead

The experiment presented in Figure 7 involved synchronization overhead measurements for an increasing number of SMPs checking in and out of the barrier. We note that the synchronization time increases linearly with the number of processes, from approximately 6.2 μs for 2 SMPs to 7.4 μs for 8 SMPs. In the current implementation all the SMPs sequentially notify an event in a single SMP, which acts as a host, upon entering the barrier. A more scalable algorithm, based on a tree structure, should be able to guarantee a logarithmic behavior for a large-scale configuration.

5.3 Unidirectional Ping

The results for the ping benchmarks are presented in Figure 8. The experiments, coded with Elan3lib, analyze the communication performance with different buffer mappings. The peak bandwidth of 335 MB/s is reached when

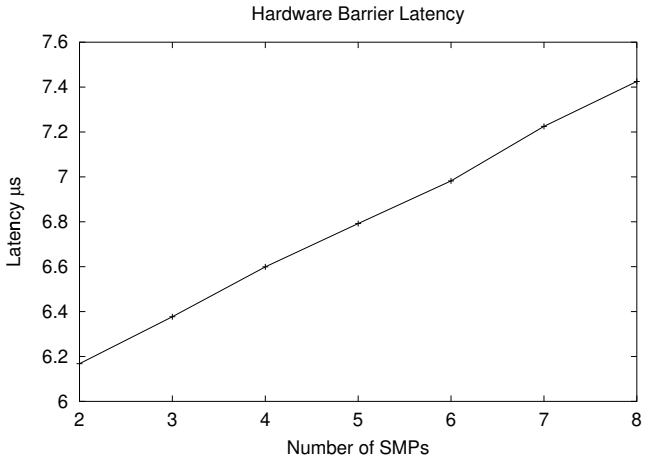

Figure 7: Barrier

both source and destination buffers are placed in the Elan memory, as shown in Figure 8 a). The maximum packet size that can be sent by the current Elan implementation is 320 bytes of data payload. This is 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.

Another important characteristic is the asymmetry in performance between the main-to-Elan and Elan-to-main mappings. This is somewhat counterintuitive, but it can be explained by an asymmetry in the PCI bus, allowing larger bandwidth for operations incoming into the Elan. The burst write performance of the PCI bus is 275 MB/s while the burst read is only 200 MB/s. This asymmetry is also present in other PCI chip-sets (e.g. ServerSet III HE and LE[6]).

The overlap between the bandwidth in the main-to-Elan and main-to-main experiments indicates that the bottleneck is in the PCI bus, for outgoing memory operations. We also note the low memory bandwidth for the memory-to-memory communications when the processors belong to the same

[6]www.serverworks.com

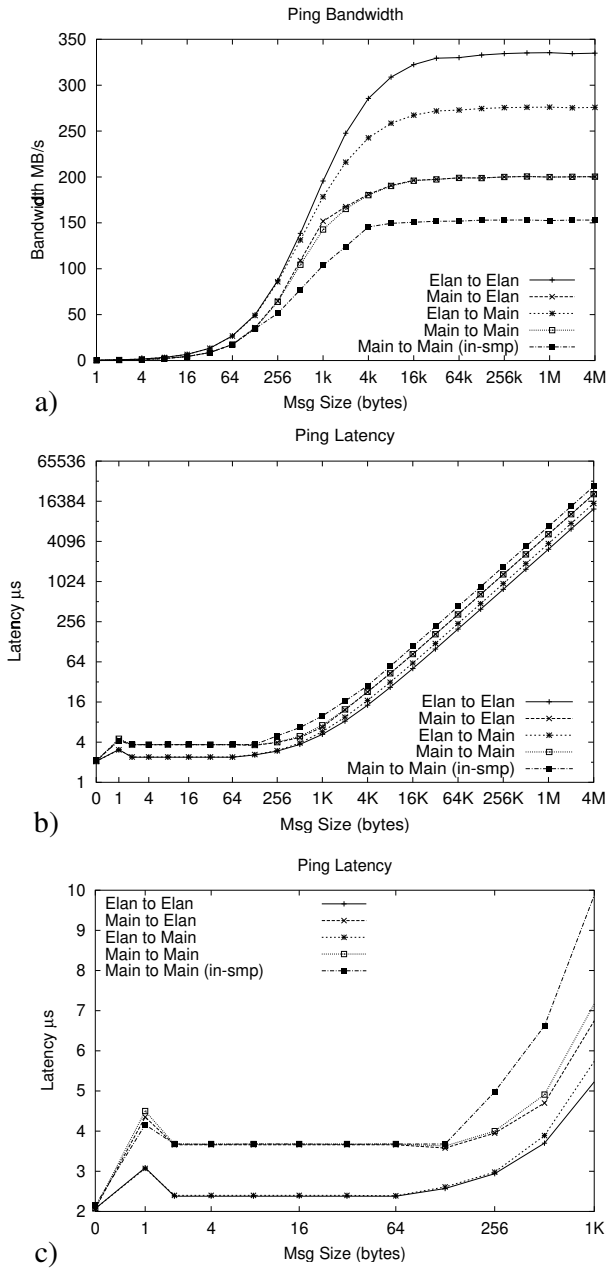

Figure 8: Unidirectional ping

SMP. This is due to the fact that these operations take place through the Elan, rather than through the memory bus within the SMP box. For all buffer mapping strategies, $N^{1/2}$ is approximately 512 bytes.

Figure 8 b) shows the latency graph, and Figure 8 c) shows a blowup of the region in the range $[0 \ldots 1KB]$. The basic latency for 0-byte messages is only 2.1 μs and is constant for messages up to 64 bytes, because these messages can be packed as a single write-block transaction. When both source and destination buffers are in the Elan, the network can deliver up to 64 bytes in 2.4 μs. Main memory to main memory transfers require 3.7 μs in the same message range. Again, memory reads issued over the PCI bus are slower then memory writes. The slight increase for 1-byte messages is due to some extra overhead to extract the single byte from the incoming message and deal with misalignment problems.

5.4 Bidirectional Ping

In this benchmark, whose results are shown in Figure 9, we continue to draw the curves corresponding to the five experiments described in the previous case. The first overall observation is that, whereas the curves in Figure 8 are very smooth, the ones in this case exhibit some fluctuations. 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, with both communication buffers mapped on the Elan is about 270 MB/s, whereas in the previous case it was 335 MB/s. This difference in the Elan-to-Elan bandwidth identifies potential bottlenecks in the network or the network interface, as opposed to the PCI bus. Potential causes of this performance degradation can be 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. We plan to further investigate these problems using the performance counters available on the Elan.

The bidirectional bandwidth for the main memory to main memory traffic is 140 MB/s. We anticipate here that the results for the total exchange will depend upon this bandwidth.

The bidirectional bandwidth for the processors inside the same SMP is low, 70-80 MB/s, compared to 150 MB/s for unidirectional experiments. This is due to the doubling of the message traffic through the Elan, since the memory traffic in this case takes place through the Elan and not through

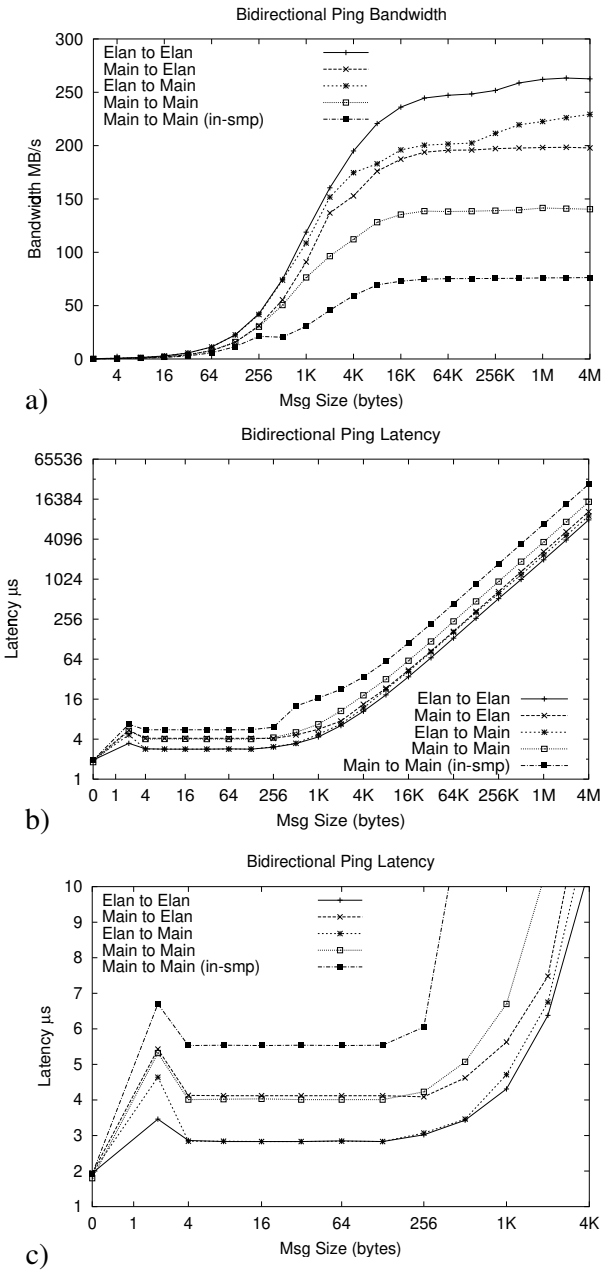

Figure 9: Bidirectional Ping

SMP memory bus. We note that is optimization is feasible and important to implement.

The bidirectional latency measurements are shown in Figure 9 b). We note (Figure 9 c) that the latency increase for small messages is about 1/2 μs: from 2.3 μs to 2.8 μs from Elan-to-Elan communication and from 3.6 μs to 4.1 for main memory to main memory. The main-to-main case for intra-SMP processors leads to higher latency due to the increased memory traffic over the PCI bus.

5.5 Read Hotspot

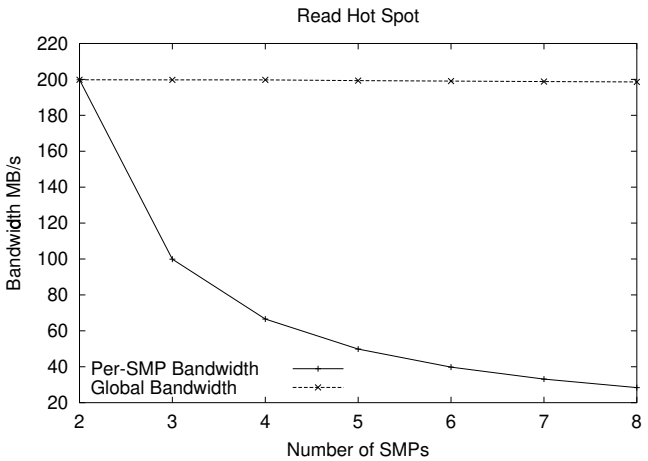

Figure 10: Read Hotspot

In this experiment we attempt to read the same memory location from an increasing number of processors (one per SMP). The bandwidth plot is depicted in Figure 10. The upper (constant) curve is the aggregate bandwidth of all processes. The curve is remarkably flat, to a constant value of 200 MB/s, exactly the value in Figure 8 a) for the main memory to main memory curve. The lower curve is 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. Hot-spot operations, both on read and on write,

are very common in scientific computing, hence the scalability of hotspot resolution is very important.

5.6 Write Hotspot

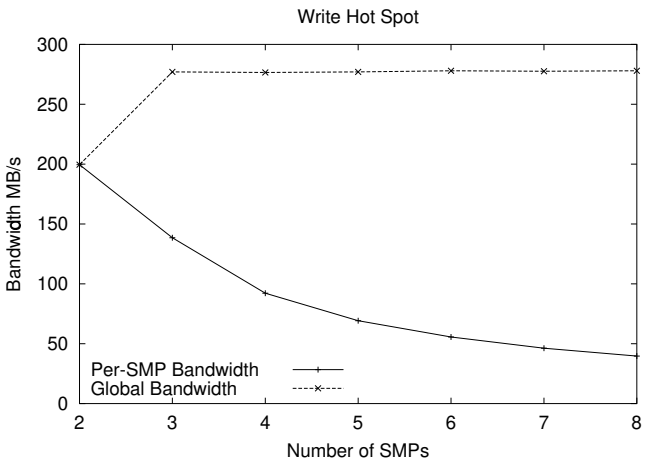

Figure 11: Write Hotspot

The two curves depicted in figure 11 are the same as those in Figure 10, only for writes. The global bandwidth on writes is pretty constant to a value of 275 MB/s, the upper bound determined by the PCI bus for the burst write. The initial jump in bandwidth when going from 2 to more processors is due to the fact that on the sending side, only 200 MB/s can be achieved. On the receiving side, as seen from figure 8 a), 275 MB/s are achievable.

5.7 Total Exchange

In this type of communication kernel, all processors are involved in both sending and receiving. As already mentioned earlier, the maximum achievable performance in this case, based on the measurements of the bidirectional bandwidth is 140 MB/s per node. We see from figure 12 a) that the maximum measured bandwidth is approximately 130 MB/s, relatively close to the

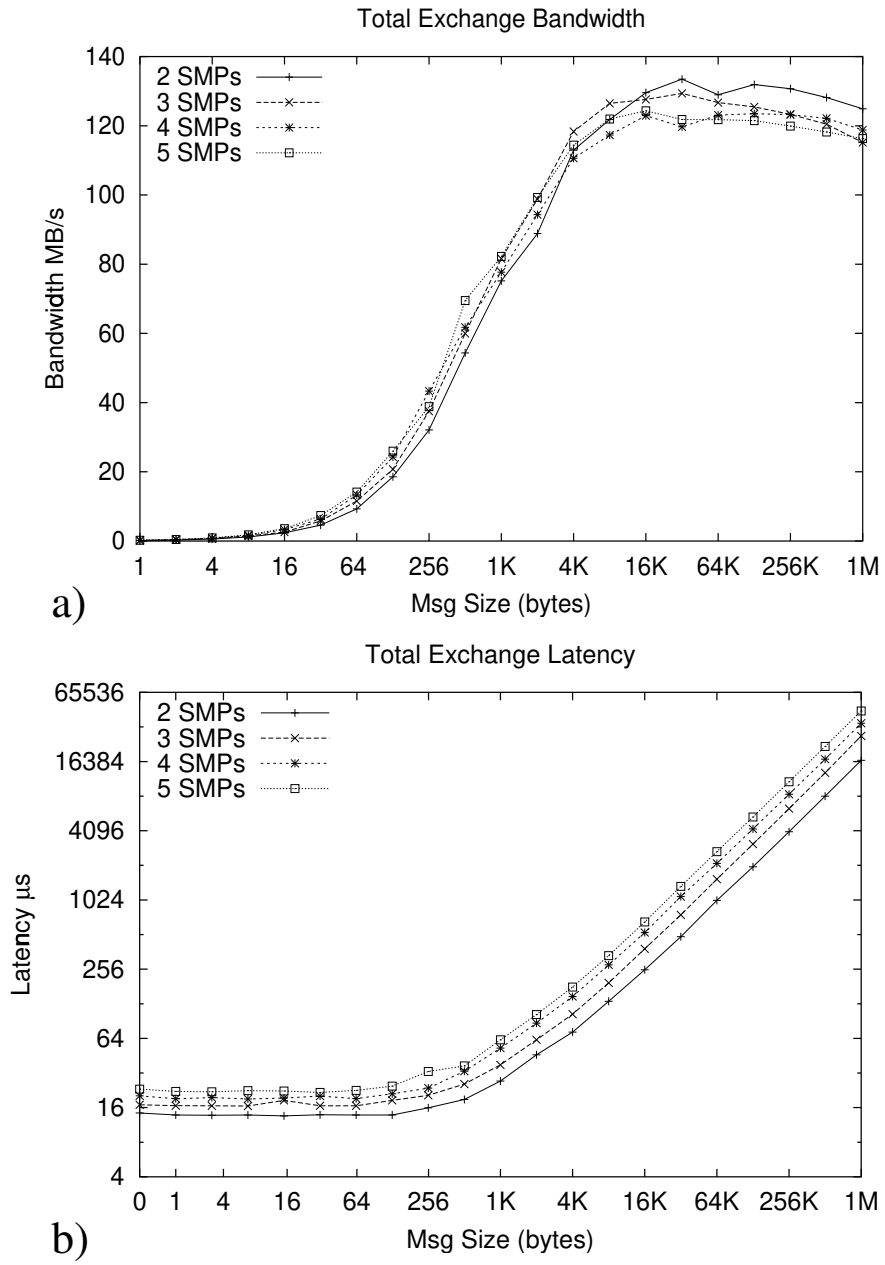

Figure 12: Total Exchange

140 MB/s achievable value. The worrisome trend clearly seen in the figure is the degradation for message sizes larger than 256 KB. This indicates an inefficiency in the buffering scheme, which would lead to scalability problems for large global communication kernels.

Figure 12 b) shows the latency for global communication. For messages up to 256 bytes, the latency is constant to a value of 20 μ. As this involves the cost of synchronization and actual message exchanges, the value is relatively low.

5.8 Elan3lib, Tports and MPI

The following set of experiments are done at the Tports and the MPI level in the communication layer pyramid. Tports is the highest level at which experiments involving communication buffers in the Elan are possible. At the MPI level, the memory allocators described in Section 3.1 are not accessible.

5.8.1 Unidirectional Ping

Figure 13 compares the performance of the unidirectional ping for the three communication libraries. The lowest values of approximately 200 MB/s for the bandwidth (the lowest two overlapped curves) are MPI and the Tports when the buffers are allocated in the main memory. At Tports level, when the source and destination buffers are allocated in the Elan memory, we note a significant bandwidth increase in all message size ranges, as expected. The saturation value of the bandwidth in this case increases from 200 MB/s to 335 MB/s. The intermediate value of 275 MB/s is obtained when the source buffer is in the Elan and the destination buffer is in main memory. An interesting observation from these measurements comes from noting that the 6 plots in the figure are grouped in 3 sets of two with identical saturation values. Given the fact that the bottleneck, as explained above, is the PCI bus, the same maximum value of the bandwidth is achieved when the source buffer is in the main memory, no matter where the destination is. However, up to the maximum rate allowed by the PCI bus, the values of the bandwidth differ, for smaller message sizes. The differences between the curves in each set is visible for a message size range between 64 bytes and 32 KB. We also note that none of these measurements is noisy.

The latency measurements for Tports and MPI, contrasted with Elan3lib are shown in figure 14. We note an increase in the latency at the Tports and MPI level, compared to the latency at the Elan3lib level, from approximately 3 μs to 5 μs. The extra cost is due to the overhead of tag matching, meaning that 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. We see that the "latency" curves further diverge for larger message sizes. The noisy upper 2 curves are the latency measurements for Tports when the source is mapped into the Elan. Currently we have no explanation for the performance glitch shown in this measurement.

5.8.2 Bidirectional Ping

We note in Figure 15 the noisier curves for the bidirectional bandwidth. Most notably, the MPI measurements indicate a sharp drop in the bandwidth in this important case for global communication kernels. As opposed to the unidirectional bandwidth measurement, the unidirectional bandwidth based on a bidirectional measurement is 50% lower for some message sizes. By contrast, the values for Tports are not lower by the same amount. The lower values are consistent with the observation that, in this case, we are hitting the maximum rate allowed by the PCI bus of 140MB/s for bidirectional bandwidth. The bidirectional bandwidth is significantly higher at the Elan3lib level, indicating that implementation of global communication kernels at this level would pay off from a performance standpoint.

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 2.1 μs and the bandwidth as high as 200 MB/s. These performance numbers are influenced by the mapping of the buffers in various levels of the memory hierarchy. The raw 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.

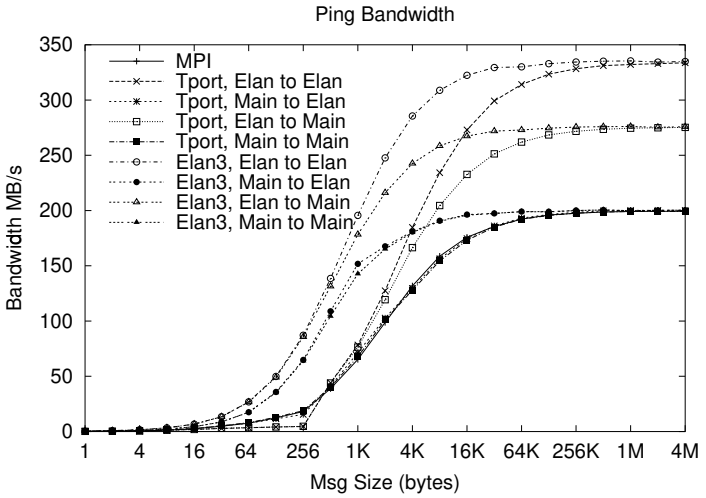

Figure 13: Unidirectional Ping Bandwidth

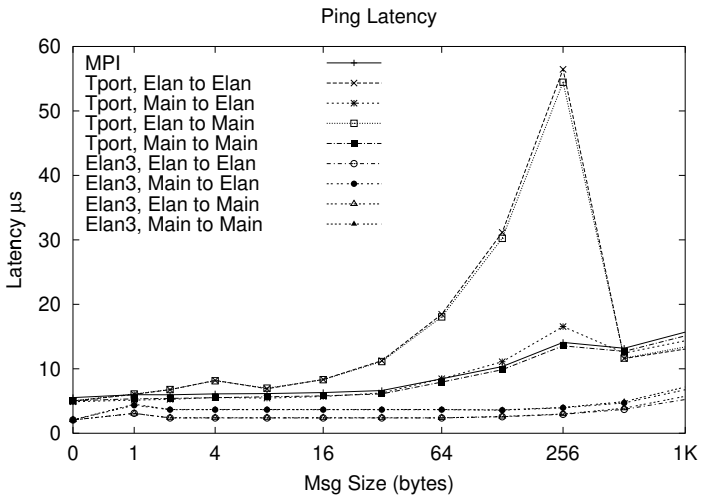

Figure 14: Unidirectional Ping Latency

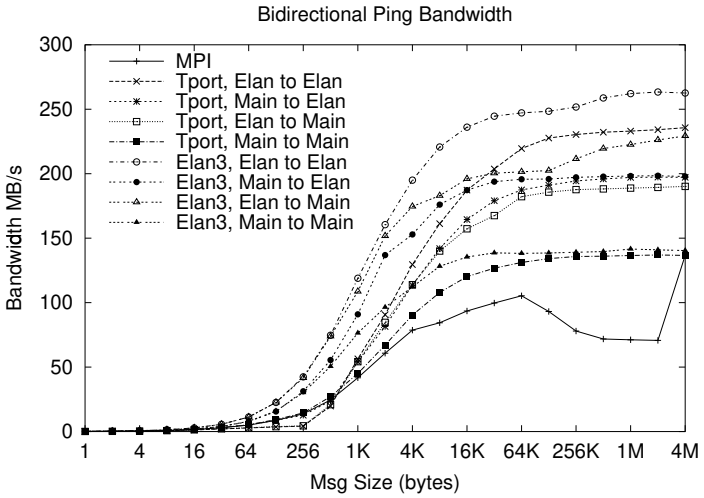

Figure 15: Bidirectional Ping Bandwidth

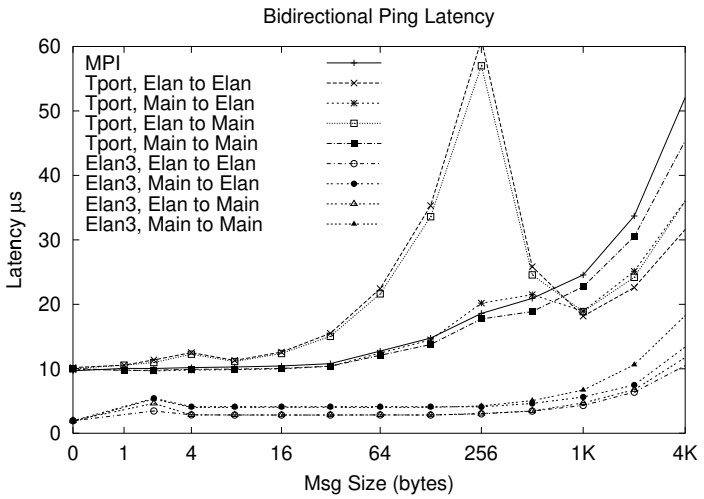

Figure 16: Bidirectional Ping Latency

At higher levels in the communication hierarchy, Tports still exhibit excellent performance figures comparable to the ones at Elan3lib level. The current MPI implementation shows some performance degradation under contention. 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. The glitches that we uncovered are likely to be fixed by the network designers in the near future.

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] J. Bruck, C. T. Ho, S. Kipnis, and D. Weathersby. Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems. In *Proceedings of the 6th ACM Symposium on Parallel Architectures and Algorithms*, pages 298–309, 1994.

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

[4] William Dally. Network and Processor Architecture for Message-Driven Computers. In R. Suaja and G. Birthwhistle, editors, *VLSI and Parallel Computers*, chapter 3, pages 140–222. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991.

[5] 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.

[6] 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.

[7] Stefan Goedecker and Adolfy Hoisie. *Performance Optimizations of Numerically Intensive Codes*. SIAM, March 2001.

[8] 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.

[9] K. D. Gunther. Prevention of Deadlocks in Packet-Switched Data Transport Systems. *IEEE Transactions on Communications*, COM-29(4):512–524, April 1981.

[10] 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.

[11] S. Hinrichs, C. Kosak, D O'Hallaron, T. M. Stricker, and R. Take. An Architecture for All-to-All Personalized Communication. In *Proceedings of the 6th ACM Symposium on Parallel Architectures and Algorithms*, pages 310–319, 1994.

[12] Adolfy Hoisie, Olaf Lubeck, and Harvey Wasserman. Scalability Analysis of Multidimensional Wavefront Algorithms on Large-Scale SMP Clusters. In *The Ninth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'99)*, Annapolis, MD, February 1999.

[13] S. L. Johnsson and C. T. Ho. Optimal Broadcsting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38:1249–1268, 1989.

[14] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, USA, 1992.

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

[16] 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.

[17] 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.

[18] 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.

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

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

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

[22] Satish Rao, Torsten Suel, Thanasis Tsantilas, and Mark Goudreau. Efficient Communication Using Total-Exchange. In *Proceedings of the 9th International Parallel Processing Symposium, IPPS'95*, Santa Barbara, CA, April 1995.

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

[24] D. B. Skillicorn, Jonathan M. D. Hill, and W. F. McColl. Questions and Answers about BSP. *Journal of Scientific Programming*, 1998.

[25] 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.

[26] Rajeev Thakur and Alok Choudary. All-to-All Communication on Meshes with Wormhole Routing. In *Proceedings of the 8th International Parallel Processing Symposium, IPPS'94*, pages 561–565, Cancun, Mexico, April 1994.

[27] Yu-Chee Tseng and Sandeep K. S. Gupta. All-to-All Personalized Communication in a Wormhole-Routed Torus. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):498–505, May 1996.

[28] 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 k-ary n-trees

In this section we formalize the fat-trees, giving a recursive definition that is general enough to embed many different topologies that are often quoted as fat-trees. We then turn our attention to a particular subclass: the k-ary n-trees. As the k-ary n-cubes and the k-ary n-butterflies [4], the k-ary n-trees are a parametric family of regular topologies that can be built varying the two parameters k and n.

Definition A.1 *A fat-tree is a collection of vertices connected by edges and is defined recursively as follows.*

- *A single vertex by itself is a fat-tree. This vertex is also the* root *of the fat-tree.*
- *If $v_1, v_2, \ldots, v_i$ are vertices and $T_1, T_2, \ldots, T_j$ are fat-trees, with $r_1, r_2, \ldots, r_k$ as roots (j and k need not to be equal), a new fat-tree is built by connecting with edges, in any manner, the vertices $v_1, v_2, \ldots, v_i$ to the roots $r_1, r_2, \ldots r_k$. The roots of the new fat-tree are $v_1, v_2, \ldots, v_i$.*

Definition A.1 is extremely general and covers many existing examples in the literature. It includes ordinary trees, "full" fat-trees with variable-sized switches and multiple connections between vertices and irregular constructions. The only exception being the orthogonal fat-trees, that allow connections between the roots $v_1, v_2, \ldots, v_i$. Some examples are shown in Figure 17.

Let's turn our attention to a particular class of fat-trees, the k-ary n-trees. k-ary n-trees borrow from a popular class of multistage interconnection networks, the k-ary n-butterflies [14] (or k-ary n-flies for short), the topology of the internal switches. The k-ary n-fly is a generalization of the butterfly and is useful to model those topologies that use communication switches with k greater than two. This topology has a recursive structure: one k-ary n-fly contains k butterflies of dimension $n-1$ as subgraphs. Also, each level 0 switch is linked to any level n switch by a unique path of length n, that is k-ary n-flies are *banyan* networks.

We can define the class of k-ary n-trees in the following way.

Definition A.2 *A k-ary n-tree is composed of two types of vertices: $N = k^n$ processing nodes and nk^{n-1} $k*k$ communication switches [7]. Each node is an n-tuple $\{0, 1, \ldots, k\text{–}1\}^n$, while each switch is defined as an ordered pair $\langle w, l \rangle$, where $w \in \{0, 1, \ldots, k-1\}^{n-1}$ and $l \in \{0, 1, \ldots, n-1\}$.*

- *Two switches*

$$\langle w_0, w_1, \ldots, w_{n-2}, l \rangle \quad \textit{and} \quad \langle w'_0, w'_1, \ldots, w'_{n-2}, l' \rangle$$

are connected by an edge if and only if $l' = l + 1$ and $w_i = w'_i$ for $i \neq l$. The edge is labelled with w'_l on the level l vertex and with w_l on the level l' vertex.

[7] A k-ary n-tree of dimension $n = 0$ is composed of a single processing node

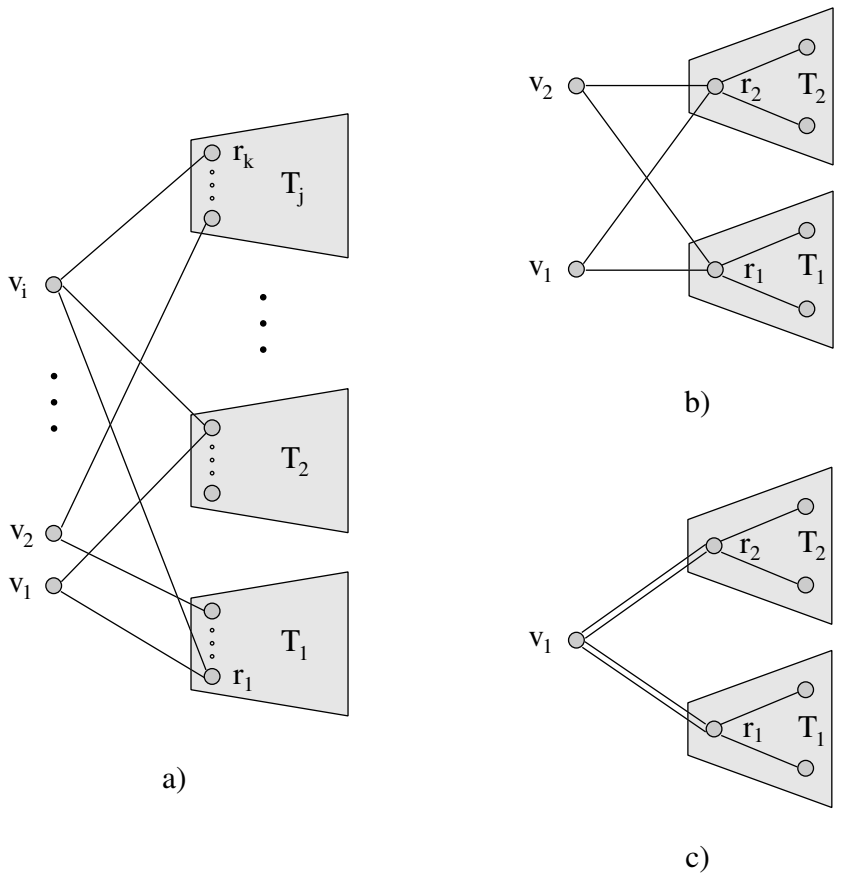

Figure 17: a) A fat-tree is recursively built connecting the new roots $v_1, v_2, \ldots, v_i$ to the roots $r_1, r_2, \ldots, r_k$ of the subtrees. b) A fat-tree with two roots. c) A fat-tree with multiple edges between the root v_1 and the roots of the subtrees T_1 and T_2.

- *There is an edge between the switch* $\langle w_0, w_1, \ldots, w_{n-2}, n{-}1\rangle$ *and the processing node* $p_0, p_1, \ldots, p_{n-1}$ *if and only if*

$$w_i = p_i \quad for \quad i \in \{0, 1, \ldots, n-2\}.$$

This edge is labelled with p_{n-1} *on the level* $n-1$ *switch.*

It can be easily seen that a k-ary n-tree is a fat-tree, according to Definition A.1. In fact, the level 0 switches $\langle w, 0\rangle$ are the roots of a k-ary n-tree whose subtrees are $(n-1)$-dimensional k-ary n-trees. Also, the labelling scheme shown in Definition A.2 makes the k-ary n-tree a delta network [14]: any path starting from a level 0 switch and leading to a given node $p_0, p_1, \ldots, p_{n-1}$ traverses the same sequence of edge labels $(p_0, p_1, \ldots, p_{n-1})$. Some examples are shown in Figure 18.

A.1 Topological properties

k-ary n-trees are built using two building blocks: processing nodes and switches. These switches are logically arranged in a $n * k^{n-1}$ matrix. Communication between a pair of nodes p and q takes place within one of the minimal subtrees of the network that contain both source and destination. The roots of these subtrees can be determined using the numerical representation of both nodes.

Definition A.3 *Given a pair of nodes*

$$p = p_0, p_1, \ldots, p_{n-1} \quad and \quad q = q_0, q_1, \ldots, q_{n-1}, \; p \neq q,$$

the minimal different index *of* p *and* q, $mdi(p,q)$, *is defined as*

$$mdi(p,q) \;=\; \min\{j \mid p_j \neq q_j\}. \tag{2}$$

From the hypothesis that $p \neq q$, we know that there is at least an index j such that $p_j \neq q_j$. The $mdi(p,q)$ is $n{-}1$ when both nodes are rooted at the same level $n{-}1$ switch, 0 when p and q belong to distinct k-ary n-trees of dimension $n{-}1$, and an intermediate value between 0 and $n-1$ otherwise. The mdi can be used to compute the minimal distance between two nodes.

Corollary A.1 *The distance between two nodes* p *and* q $p \neq q$, $d(p,q)$, *is given by*

$$d(p,q) = 2\,(n - mdi(p,q))\,. \tag{3}$$

Also, we can define the minimal different index of a node and a switch.

Definition A.4 *Given a node* p *and a switch* s

$$p = p_0, p_1, \ldots, p_{n-1}, \quad s = \langle w_0, w_1, \ldots, w_{n-2}, l\rangle,$$

the minimal different index *of* p *and* s, $mdi(p,s)$, *is defined as*

- $n-1$ *if* $p_i = w_i,\ i \in \{0, 1, \ldots, n-2\}$,
- $\min\{j \mid p_j \neq w_j\}$ *otherwise.*

As in the previous case, the minimal different index between a node and a switch is a number in the range $\{0, 1, \ldots, n–1\}$. Using the minimal different index we can characterize the set of nearest common ancestors of any pair of nodes.

Definition A.5 *Given two nodes* $p = p_0, p_1, \ldots, p_{n-1}$ *and* $q = q_0, q_1, \ldots, q_{n-1}$, $p \neq q$, *with minimal different index* $m = mdi(p, q)$, *the set of their* nearest common ancestors $nca(p, q)$ *can be defined as*

$$nca(p, q) \;=\; \{\langle p_0, p_1, \ldots, p_{m-1}, w_m, w_{m+1}, \ldots, w_{n-2}, m\rangle\}, \tag{4}$$

with $w_i \in \{0, 1, \ldots, k-1\}$.

The cardinality of $nca(p, q)$ is $k^{n-1-mdi(p,q)}$ and varies between 1, when the nodes are directly connected to the same switch, and k^{n-1}. In the second case, $nca(p, q)$ is the set of all level 0 switches.

In the worst case, a message sent from node p to node q must reach one of the level 0 switches and then follow the only path to the destination. This implies that the diameter D of this network is

$$D \;=\; 2n \;=\; 2\log_k N. \tag{5}$$

To compute the average distance d_m we can observe that each node has $k-1$ nodes at distance 2, $k^2 - k$ nodes at distance 4, ..., $k^n - k^{n-1}$ nodes at distance $2n$. The solution of the corresponding geometric series gives

$$d_m \;=\; 2\left(n - \frac{1}{k-1}\right) + \frac{2}{k^n(k-1)} \;\approx\; 2\left(n - \frac{1}{k-1}\right). \tag{6}$$

This analytic formulation shows that the average distance is very close the network diameter. For example, in the 4-ary n-trees $d_m \approx 2n - 2/3$.

When a k-ary n-tree is divided into two symmetric halves, $k^n/2$ links between level 0 and level 1 switches are cut. Thus, the bisection width B is

$$B \;=\; \frac{k^n}{2} \;=\; \frac{N}{2}. \tag{7}$$

As in the butterflies, the bisection width scales as a linear function of the number of nodes. Also, the overall communication bandwidth between level i and level $i+1$, $i \in \{0, 1, \ldots, n–2\}$ and between level n–1 and the processing nodes is constant and proportional to the number of nodes N.

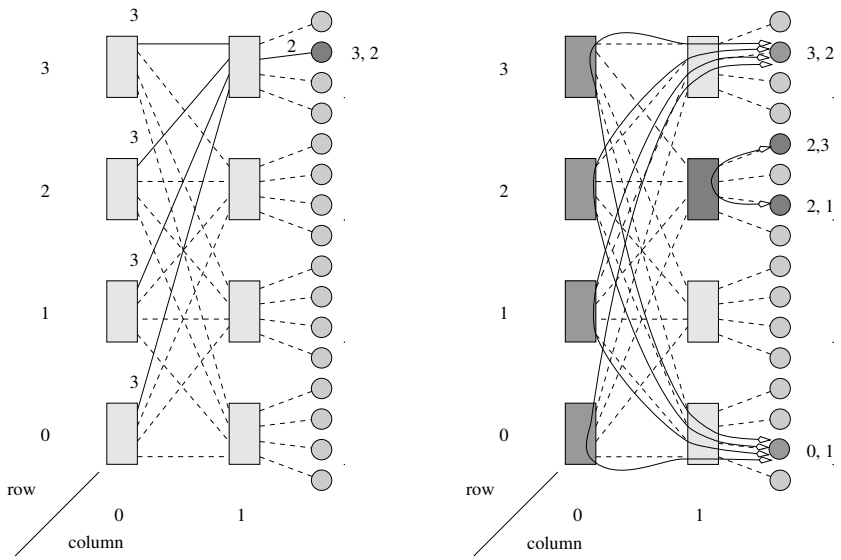

Figure 18: a) The link labelling of Definition A.2 makes the k-ary n-tree a delta network: all paths connecting level 0 switches to the node $3, 2$ (solid lines) cross the same sequence of edge labels $(3, 2)$. b) The *mdi* of nodes $2, 1$ and $2, 3$ is 1: this implies that their *nca* is the common root $\langle 2, 1 \rangle$. The *mdi* of nodes $0, 1$ and $3, 2$ is 0; in this case the *nca* is composed of all level 0 switches. These two nodes are connected by four distinct minimal paths.

A.2 Message routing

As outlined above, minimal routing between a pair nodes on a k-ary n-tree can be accomplished sending the message to one of the nearest common ancestors and from there to the destination. That is, each message experiences two phases, an *ascending* phase to get to a nearest common ancestor, followed by a *descending* phase. If we attach at the beginning of each message a header containing the address of the destination $p = p_0, p_1, \ldots, p_{n-1}$, the switches can execute straightforward routing algorithms using the edge labelling scheme of Definition A.2. Each switch $s = \langle w_0, w_1, \ldots, w_{n-2}, l \rangle$ has $2k$ edges: k of these are connected to level $l-1$ switches (if $l > 0$) and the remaining k to level $l+1$ switches or to processing nodes. We will call, the former, level $l-1$ edges and, the latter, level l edges. The following fragment of pseudo-code describes the skeleton of such a class of routing algorithms.

---

if (the message comes from a level l edge)
    /* the message is in the ascending phase */
    if ($mdi(p, s) = l$)
        /* begin the descending phase */
        ⟨ route the message to the level l edge labelled p_l ⟩
    else
        /* continue the ascending phase */
        ⟨ pick one of the level $l-1$ edges ⟩
else
    /* the message is in the descending phase */
    ⟨ route the message to the level l edge labelled p_l ⟩

Figure 19: The skeleton of the routing algorithms.

The routing decision is taken according to the provenience of the message: if it comes from a level l edge, the message is in the ascending phase. The switch reads the message header and computes the $mdi(p, s)$: if it is equal to l, the switch is a nearest common ancestor according to Definitions A.3 and A.5 and the message can begin the descending phase. The outgoing edge is chosen according to the labelling scheme of Definition A.2, in order to correct the l^{th} field. Otherwise, the message continue its ascending path: any of the level $l-1$ edges will do. According to the way we choose this edge we can have different routing algorithms.

- A *deterministic* algorithm always chooses the same path for a given pair of nodes.
- An *adaptive* algorithm makes a decision according to the local state of the switch,

avoiding congested edges. This version is expected to give the best performance, at the cost of a major complexity in the implementation.

Descending messages, coming from a level l–1 edge, are routed, according to the usual labelling scheme, to the only path leading to the destination.

The deadlock-freedom[8] can be easily proved building an acyclic buffer or channel dependency graph, according to the flow control strategy in use [9] [5].

[8]The algorithms are livelock-free because they are minimal and so, they guarantee the progress of each message toward the destination at each routing step.