Assessing MPI Performance on QsNet^{II}

Pablo E. García[1], Juan Fernández[1],
Fabrizio Petrini[2], and José M. García[1]

[1] Departamento de Ingeniería y Tecnología de Computadores,
Universidad de Murcia, 30071 Murcia, Spain
{pablo.garcia, juanf, jmgarcia}@ditec.um.es
[2] CCS-3 Modeling, Algorithms & Informatics,
Los Alamos National Laboratory, Los Alamos, NM 87545, USA
fabrizio@lanl.gov

Abstract. To evaluate the communication capabilities of clusters, we must take into account not only the interconnection network but also the system software. In this paper, we evaluate the communication capabilities of a cluster based on dual-Opteron SMP nodes interconnected with QsNet^{II}. In particular, we study the raw network performance, the ability of MPI to overlap computation and communication, and the appropriateness of the local operating systems to support parallel processing. Experimental results show a stable system with a really efficient communication subsystem which is able to deliver 875 MB/s unidirectional bandwidth, 1.6 μsec unidirectional latency, and up to 99.5% CPU availability while communication is in progress.

1 Introduction

Clusters have become the most successful player in the high-performance computing arena in the last decade. At the time of this writing, many of the fastest systems in the Top500 list [14] are clusters. These systems are typically assembled from commodity off-the-shelf (COTS) components. In particular, there is a growing interest in those systems assembled with SMP nodes based on 64-bit processors –mainly Itanium2 and Opteron– interconnected with high-performance networks, such as Infiniband [11], Myrinet [12] or Quadrics [13].

Performance of large-scale clusters is determined by the parallel efficiency of the entire system as a whole rather than by the peak performance of individual nodes. In order to achieve a high degree of parallel efficiency, there must be a proper balance over the entire system: processor, memory subsystem, interconnect, and system software. If we focus on the communication capabilities, we must pay attention not only to the interconnection network but also to the communication system software. In this case, while Myrinet, Infiniband and Quadrics are the preferred choices for the cluster interconnect, MPI [10] is the *de facto* standard communication library for message-passing, and Linux is the most popular choice for operating the cluster nodes.

The traditional approach to evaluate the communication capabilities of a cluster primarily relies on bandwidth and latency tests. Even though the bandwidth and latency figures are significant, they are not enough to characterize

B. Di Martino et al. (Eds.): EuroPVM/MPI 2005, LNCS 3666, pp. 399–406, 2005.

the system behavior when running scientific and engineering applications in a cluster. There are other aspects that have a great impact in the communication performance as well. On the one hand, the ability of the MPI layer to overlap communication and computation may limit the CPU availability for user applications. An efficient MPI implementation should leverage modern network interface cards to offload protocol processing. On the other hand, the commodity OSes running in every node may interfere with user-level processes. The nodes of a cluster should be properly tuned in order to minimize the computational noise introduced by unnecessary dæmons, services, and tools. In this paper, we focus on analyzing all these aspects on a cluster based on dual-Opteron SMP nodes interconnected with QsNet^{II} from Quadrics.

The rest of the paper is organized as follows. The next section presents the main features of QsNet^{II}. In particular, both the Elan4 network interface card and the Elite4 switch are described. In Section 3, different performance aspects of a QsNet^{II}-based cluster assembled with Opteron nodes are analyzed. Finally, we conclude with some remarks and an outline of future work.

2 QsNet^{II}

QsNet from Quadrics have become one of the preferred choices to interconnect large-scale clusters since its appearance in 1996 [14]. This success is due to the fact that QsNet provides low-latency and high-bandwidth interprocessor communication through a standard interface for systems based on commodity processing nodes. The latest version, QsNet^{II}, was released in 2004. Some large-scale systems based on QsNet^{II}, such as *Thunder* [14], have already been deployed.

QsNet^{II} consists of two ASICs: Elan4 and Elite4. Elan4 is the core for the QsNet^{II} network interface card (NIC). The Elan4 NICs connect commodity processing nodes to the QsNet^{II} network through a standard interface, PCI-X. In turn, Elite4 is capable of driving eight bidirectional links at 1.3 GB/sec each way. The Elite4 switches form a multistage network to interconnect the Elan4 NICs attached to the processing nodes. These are the most salient aspects of QsNet^{II}:

- 64-bit architecture. Both the Elan4 and the Elite4 components have an internal 64-bit architecture and fully support a 64-bit virtual address space.
- PCI-X interface. Elan4 NICs implement a 64-bit, 133 MHz, PCI-X interface.
- DMA engine. User processes can perform read/write from/on remote memory locations by just issuing Remote DMA (RDMA) commands to the Elan4.
- Event engine. Events are used for synchronization purposes and available to the Elan4 processors and the main CPU. Events are triggered upon completion of communication operations (e.g. RDMA transactions).
- Virtual operation. QsNet^{II} extends the conventional virtual memory mechanism so that user processes can transfer data directly between their virtual address spaces.
- Programmability. In addition to the internal command processor, the Elan4 NICs provide a 64-bit RISC programmable thread processor.

- Support for collectives. QsNet^{II} provides hardware-supported multicast operations over subnets of nodes to cut down the synchronization time.
- Reliability. QsNet^{II} implements a packet-level link protocol in hardware which is able to detect faults, route packets around faulty areas, and even retransmit lost packets. Moreover, packets are CRC-protected to detect data corruption.

2.1 Elan4

The Elan4 NICs are in charge of injecting and receiving packets into and from the network. In addition, every Elan4 NIC incorporates a programmable thread processor to offload higher-level protocol processing to the NIC. The Elan4 NIC functional units are interconnected using several separate 64-bit data buses to increase concurrency and reduce latency operation. The main Elan4's functional units are the *command processor*, the *thread processor*, the *DMA engine*, the *event engine*, and the *Short Transaction Engine* (STEN). The *command processor* processes commands from either the main CPU or other Elan4's functional units. Under this model, there is a command port mapped directly into the user process's address space. Each command port is no more than a command queue where user processes can directly issue one or more commands to the Elan4 without OS intervention. In this way, the command processor executes commands from different user processes on their behalf. Also, the command processor controls the thread processor, the DMA engine, the event engine and the STEN. The *thread processor* is a 200 MHz, 64-bit RISC programmable thread processor enriched with special instructions to support lightweight threads. This thread processor can be programmed in C and is used to aid the implementation of communication libraries without explicit intervention from the main CPU. The *Short Transaction Engine* (STEN) is closely integrated with the command processor. This specialized functional unit is optimized to handle short messages. For further details about Elan4 see [5].

2.2 Elite4

The Elite4 is an eight-port crossbar, with two virtual channels per link, that can deliver 1.3 GB/sec each way. QsNet^{II} connects the Elite4 switches in a quaternary fat-tree topology. The Elite4 switches use source routing to implement an *up/down* routing algorithm which takes advantage of this topology. The routing tags can identify either a single output link or a group of links for multicast transfers. The routing algorithm is adaptive in the *up* phase and deterministic in the *down* phase. The implementation of this routing algorithm is highly efficient and introduces a delay of approximately 20 *ns* per switch.

At the link level, packets are divided into smaller 32-bit flits to use wormhole flow control. Every packet transmission creates a virtual circuit between the source and the destination node. The virtual circuit is closed after the destination node acknowledges packet reception.

3 Performance Evaluation

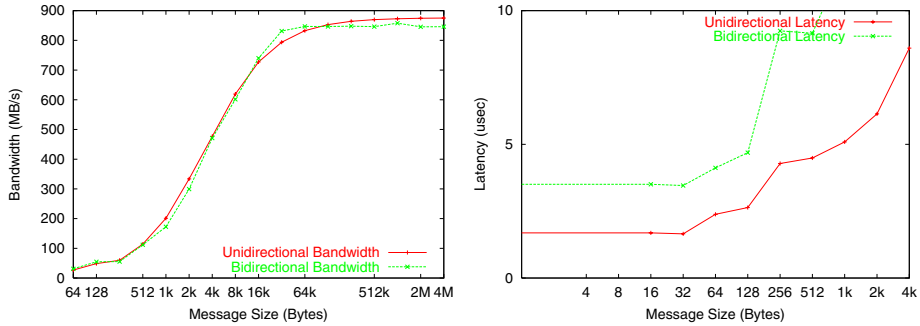
In this section, we present the performance results obtained in our initial evaluation of a cluster based on dual-Opteron SMP nodes interconnected with QsNet^{II}. In particular, we have conducted experiments to measure the basic network performance , the ability of QsNet^{II}'s MPI to overlap computation and communication, and the level of intrusiveness of the local OS in user-level computation. Table 1 summarizes the experimental setup.

Table 1. Experimental Setup

Characteristic		Description
Nodes	Processor	2xAMD Opteron 244 1.8 GHz
	Chipset	AMD-8131 HyperTransport
	I/O Bus	64-bit PCI-X (66, 100, and 133 MHz)
	BIOS	AMIBIOS 08.00.10
	Memory	1x1GB DDR400
Interconnect	NIC	QM500b A02 PCI-X Elan4
	Switch	QS8A B01 8-port Elite4
Software	Kernel	2.4.21-178.x86_64 / 2.4.21-4.16qsnet
	OS	SuSE Linux 9.0 (x86-64)
	Libraries	qsnet2libs-1.6.9-0 / qsnetmpi-1.24-37
	Compiler	gcc-3.3.1-23
	Launcher	SLURM 0.3.8-1

3.1 Network Performance

To expose the network performance of QsNet^{II} as seen by parallel applications, we wrote our microbenchmarks at the MPI level. We perform two different experiments to characterize the network in terms of latency and bandwidth. Unidirectional bandwidth and latency are computed using a simple microbenchmark where two processes residing in two different nodes exchange messages. In this case, both processes invoke alternatively MPI_Send and MPI_Receive operations in a loop for different message sizes. In turn, bidirectional bandwidth and latency are obtained using a similar experiment where both processes send and receive messages simultaneously using MPI_Isend and MPI_Irecv operations. Figure 1(a) shows the MPI unidirectional and bidirectional bandwidth. The peak unidirectional bandwidth, obtained as half of the measured bidirectional traffic, is 875 MB/s, whereas in the bidirectional case is 857 MB/s. Figure 1(b) shows the MPI unidirectional and bidirectional latency. The minimum achievable latency is 1.58μsec for unidirectional traffic and 3.41μsec for bidirectional traffic. This remarkable low latencies are made possible by the QsNet^{II} software infrastructure which provides zero-copy, user-level network access. Finally note that these results indicate that Quadrics has improved the internal design of QsNet^{II} over the previous version of QsNet which showed a significant gap between unidirectional and bidirectional figures [7]. This improvement is due to the fact that Elan4 incorporates several separate 64-bit data buses [5].



(a) Bandwidth (b) Latency

Fig. 1. Network performance

3.2 Overlapping Computation and Communication

Network interface cards for modern cluster interconnects, such as Myrinet [1] or Quadrics [7], provide programmable processors and substantial memory. This trend opens a wide range of design possibilities for communication protocols since this added capability allows the host processor to delegate certain tasks to the NIC [9]. This offloading of protocol processing has two significant benefits. First, moving communication protocol processing to the NIC increases the availability of the host processor for use by application programs, that is, to overlap computation and communication. Second, NIC-based collectives show dramatically reduced latency and increased consistency over host-based versions when used in large-scale clusters [6].

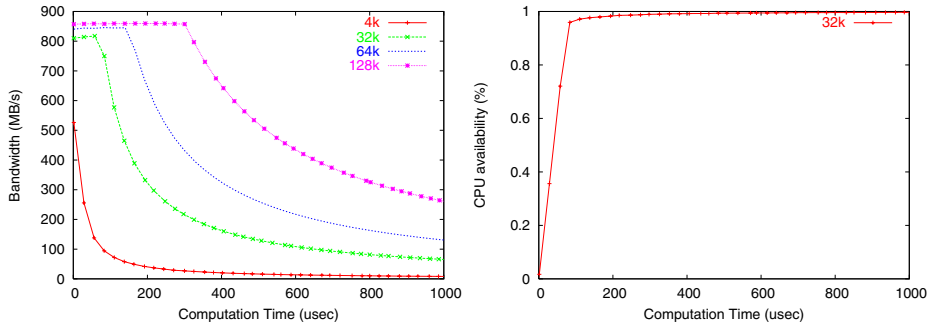
In this section, we measure the ability of QsNet^{II}'s MPI to overlap computation and communication. This capability is influenced not only by the characteristics of the underlying network, but also by the quality of the MPI implementation. In order to characterize the overlapping of computation and communication, we measure the processor availability –how much the processor is available to application programs– while communication is in progress. To do so, we use a test, namely *post-compute-wait* test [4], which combines computation and MPI communication.

Post-Compute-Wait Test. This test consists of a *worker process* and a partner *support process* running on two separate nodes. The *worker process* posts a non-blocking send and a receive directed to the partner process, performs some parametric amount of computation, and waits for the pending send and receive calls to complete. The *support process* posts a matching send and receive. The *worker process* code is instrumented to time the non-blocking call phase, the compute phase, and the wait phase. The compute phase is a do-nothing loop which keeps the host processor busy, without timers or system calls, for a predefined amount of time. This loop performs neither memory accesses nor I/O in order to avoid operations which might introduce non-determinism in the

experiments. Using this test, we have conducted several experiments in order to obtain (i) the maximum achievable bandwidth given specific message sizes and computational granularities and (ii) the CPU availability figure defined as the ratio between the total compute phase time and the total execution time.

CPU Availability and Bandwidth. In figure 2(a), we show the maximum achievable bandwidth when we increased the computational granularity up to 1 ms for four different message sizes. As expected, when the computation time is shorter than message latency, the sustained bandwidth is very close to the maximum achievable bandwidth as depicted in figure 1(a). In turn, figure 2(b) shows the CPU availability for 32 KB messages when we increase the computational granularity up to 1 ms. Note that, in this case, as soon as the computation time exceeds the roundtrip message latency, the CPU availability is about 95.9% and grows up to 99.5%.

Finally, it is worth noting that this test provides an additional check of whether the MPI library complies with the progress rule of the MPI standard. This rule determines that the non-blocking send and receive calls must complete independently of a process making MPI library calls. As we have shown, QsNet^{II}'s MPI perfectly complies with this rule.



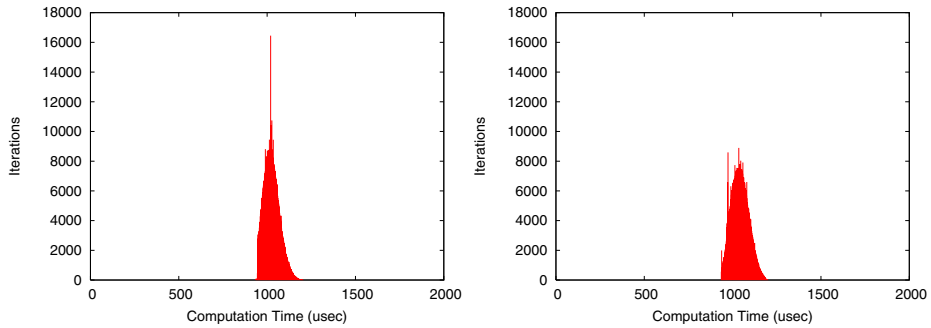
(a) Bandwidth (b) CPU availability

Fig. 2. Post-compute-wait test

3.3 Computational Noise

Performance of many parallel applications is limited by the ability of the entire system to globally synchronize all nodes. However, local operating systems running on the cluster nodes lack global awareness about parallel applications. Local OS kernels and system dæmons are randomly scheduled across cluster nodes. This unpredictable behavior has a significant impact on tightly-coupled applications in which activities on the compute nodes are highly synchronized. Moreover, this performance bottleneck get worse as cluster size increases [8].

Several techniques have been proposed in the literature to minimize the impact of system activities on the overall performance of a parallel application.



(a) Node 0, Processor 0 (b) Node 0, Processor 1

Fig. 3. Computational noise test

On the one hand, several authors have proposed different co-scheduling schemes to synchronize system activities across cluster nodes [2,3]. On the other hand, performance may be improved by just measuring the computational noise due to periodic system activities which may harm performance in order to remove them or ameliorate their impact [8]. The use of the above mentioned coscheduling techniques is not commonly used, since it requires kernel-level modifications. Therefore, in this section, we follow the second approach. We use a simple microbenchmark which quantifies computational noise due to system activities. In this microbenchmark, each node performs 1 million iterations of a synthetic computation which performs neither memory accesses nor I/O. Each synthetic computation has been calibrated to take about 1 ms in the absence of noise, that is, the run time for each iteration should always be the same in a noiseless machine.

In figures 3(a) and 3(b), we show the results for the first and the second processor on the master node, respectively[1]. From these results we can derive two interesting conclusions. First, the nodes of our cluster are noiseless even in the worse case which corresponds to the master node. Second, the distribution for the second processor has the very same shape but it is slightly displaced to the right on both nodes. This indicates that some kind of system activity is taking place only in the second processor. In this case, the only candidate is the kernel which might be descheduling the microbenchmark process in the second processor more often.

4 Conclusions and Future Work

In this paper, we have presented our initial analysis of the communication capabilities for a cluster based on dual-Opteron SMP nodes interconnected with QsNet^{II}. Our experimental results show that this platform has an extraordinary potential for high-performance cluster computing. The communication subsystem provides can rely on an extremely efficient network, a high degree of

[1] Results for slave nodes are similar and we omit them in the sake of brevity.

overlapping between computation and communication, and a really low level of computational noise due to system software interference.

Future work include scalability analysis for larger configurations, performance analysis for different traffic patterns, and study of different scenarios to offload protocol processing to the Elan4 NIC.

References

1. Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.
2. Juan Fernández, Eitan Frachtenberg, and Fabrizio Petrini. BCS-MPI: A New Approach in the System Software Design for Large-Scale Parallel Computers. In *Proceedings of IEEE/ACM Conference on SuperComputing*, Phoenix, AZ (USA), November 2003.
3. Terry Jones, William Tuel, and Brian Maskell. Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System. In *Proceedings of IEEE/ACM Conference on SuperComputing*, Phoenix, AZ (USA), November 2003.
4. William Lawry, Christopher Wilson, and Arthur B. Maccabe. COMB: A Portable Benchmark Suite for Assessing MPI Overlap. In *Proceedings of IEEE International Conference on Cluster Computing*, Chicago, IL (USA), September 2002.
5. Quadrics Supercomputers World Ltd. *Elan4 Reference Manual*.
6. Adam Moody, Juan Fernández, Fabrizio Petrini, and Dhabaleswar K. Panda. Scalable NIC-Based Reduction on Large-Scale Clusters. In *Proceedings of IEEE/ACM Conference on SuperComputing*, Phoenix, AZ (USA), November 2003.
7. Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January/February 2002.
8. Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8192 Processors of ASCI Q. In *Proceedings of ACM/IEEE Conference on SuperComputing*, Phoenix, AZ (USA), November 2003.
9. Piyush Shivam, Pete Wyckoff, and Dhabaleswar K. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *Proceedings of IEEE/ACM Conference on SuperComputing*, Denver, CO (USA), November 2001.
10. Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, 1996.
11. www.infinibandta.org. Infiniband Trade Association.
12. www.myri.com. Myricom, Inc.
13. www.quadrics.com. Quadrics Supercomputers World Ltd.
14. www.top500.org. Top500 Supercomputing Sites.