Exploiting Spatial Parallelism in Ethernet-based Cluster Interconnects

Stavros Passas[†], George Kotsis[†], Sven Karlsson[*], and Angelos Bilas[†]
Institute of Computer Science (ICS)
Foundation for Research and Technology - Hellas (FORTH)
P.O. Box 1385, Heraklion, GR-71110, Greece
{stabat,kotsis,svenka,bilas}@ics.forth.gr

Abstract

In this work we examine the implications of building a single logical link out of multiple physical links. We use MultiEdge [12] to examine the throughput-CPU utilization tradeoffs and examine how overheads and performance scale with the number and speed of links. We use low-level instrumentation to understand associated overheads, we experiment with setups between 1 and 8 1-GBit/s links, and we contrast our results with a single 10-GBit/s link. We find that: (a) Our base protocol achieves up-to 65% of the nominal aggregate throughput. (b) Replacing the interrupts with polling significantly impacts only the multiple link configurations, reaching 80% of nominal throughput. (c) The impact of copying on CPU overhead is significant, and removing copying results in up-to 66% improvement in maximum throughput, reaching almost 100% of the nominal throughput. (d) Scheduling packets over heterogeneous links requires simple but dynamic scheduling to account for different link speeds and varying load.

1. Introduction

Over the last few years, interconnects for scalable systems have been using custom architectures that achieve (a) low-latency and high-throughput, (b) low CPU utilization, and (c) scalability to large numbers of processors. Previous research has focused on building interconnects that are decoupled from host processors and memory, and has resulted in dramatically reducing the cost of such interconnects. The most common type of high performance compute system today is a cluster. 81.2% of the 500 most powerful computers are clusters including the fourth and the fifth fastest systems in the world [20]. Such clusters are normally designed by interconnecting high-end PCs with a high-performance network, such as Myrinet [6], Infiniband [1], or Quadrics [16].

However, the cost of the interconnect remains a high proportion of the system cost. This is particularly true, if we take into account maintenance costs as well. For this reason, a large number of scalable systems are using lower-cost interconnects, such as Gigabit Ethernet. 66.5% of the cluster systems in the Top500 list use off-the-shelf Gigabit Ethernet as their only interconnect.

In both cases it becomes important to examine how they can transparently benefit from spatial parallelism for increasing throughput. Both types of networks have mostly been using byte-level parallelism to increase throughput: Network links may actually consist of multiple physical links that are bundled together and each packet to be transmitted is sliced over all physical links. We refer to this as spatial (byte-level) parallelism. Both types of network are starting to use the same physical link technology at 2.5, 5, or 10 GBit/s. The main difference between cluster and more tightly-coupled interconnects is the support provided from the core of the interconnect (switches), such as flow control and reliable transmission. In this work we choose to use Ethernet-based interconnects to investigate this issue, as we believe that they have the potential for significantly improving the cost-effectiveness of scalable systems.

Our previous work [12] investigated issues related to building *edge-based* communication protocols by building *MultiEdge*, a communication protocol for Ethernet networks that:

- (a) Does not require support from the network switches for achieving in-order delivery and flow-control.

- (b) Transparently uses multiple links and offers a novel API for mixed in-order and out-of-order message delivery.

In this work we further focus on (b). Spatial parallelism is emerging as a main trend in building future scalable sys-

---

tems. Similar to multicore CPUs that try to exploit spatial parallelism at the system node level, there is a need to understand and explore spatial parallelism at the interconnection network and communication protocol level. Using multiple physical links (or end-to-end paths) for building high-speed logical links can lead to improvements in system cost-effectiveness: it will allow building scalable systems out of lower-cost components that are able to better follow technology trends.

The goal of this paper is to examine and understand overheads in building high-throughput logical links by using between 1 and 8 1-GBit/s links. We also contrast this approach to using a single 10-GBit/s link. In our experiments we use two systems connected back to back and a set of microbenchmarks. We examine how overheads scale at the host-to-link interface in terms of throughput, and CPU overhead. We investigate the impact of copying, interrupts, and packet scheduling over multiple links.

We find that using multiple physical links is limited by interrupt overhead compared to a single high speed link. Replacing interrupts with polling results in similar maximum throughput (800 MBytes/s) in 8x1 and 1x10 configurations, limited by memory copies. Artificially removing protocol copies results in achieving 100% of nominal throughput with the one-way test in all link configurations. Finally, a simple but dynamic link scheduler is required for achieving maximum throughput with heterogeneous links.

The rest of the paper is organized as follows. Section 2 provides an overview of *MultiEdge* as well as our extensions for the purposes of this work. We present and discuss our experimental platform and results in Sections 3 and 4. We discuss related work in Section 5. Finally we summarize our work and draw conclusions in Section 6.

2. Communication Protocol

In this section we briefly summarize the design and implementation of *MultiEdge* and we discuss our extensions and optimizations for handling multiple physical links. A detailed description of *MultiEdge* appears in [12].

MultiEdge is a light-weight communication subsystem for taking advantage of (i) reducing the need of protocol support from the network core and (ii) spatial parallelism. The basic communication primitives in *MultiEdge* are efficient remote read and write operations that make use of DMA transfers.

MultiEdge supports reliable transfers and deals with flow control at the edge of the system. *MultiEdge* aims at providing transparent communication semantics for both compute and storage application, and for this purpose is currently designed for and implemented in the OS kernel, requiring a system call for reading and writing remote memory. Currently, *MultiEdge* is implemented completely in software,

however, future work should examine different divisions of the protocols between the host OS and the network interface card (NIC). Hence, currently, *MultiEdge* can make use of commodity Ethernet NICs and switches that do not provide any hardware support for protocol offloading. In this paper, we modify and tune *MultiEdge* in the following ways:

- We add support for up-to 32 Ethernet links, limited by the atomic bit operations we use. This requires mostly changes to the control interfaces between *MultiEdge* and the Linux kernel.

- We artificially eliminate the user-to-kernel and kernel-to-user data copy on the send and receive paths respectively to quantify the impact of data copying on CPU overhead at high link speeds. The elimination of data copies is done by removing the corresponding sections of code in the protocol.

- We design a version of *MultiEdge* that uses polling instead of interrupts to quantify the impact of interrupt handling. *MultiEdge*'s scheme for interrupt handling has already been optimized in [12] to support hardware and software coalescing, however, it still incurs significant overheads. To implement the polling mechanism, we modify *MultiEdge*'s receive path to poll each link for transmission completions and incoming packets.

- We examine the impact and trade-offs between three different simple link-level schedulers:

  1. Static round robin (SRR): schedules one packet per link in a round-robin fashion, without any further knowledge. This is the default scheduler used in [12].

  2. Weighted static round robin (WSRR), where each link has a static weight (W) which indicates its peak, relative with the other links, speed. The WSRR schedules W frames per link before switching to the next link in a round-robin fashion.

  3. Weighted dynamic (WD), where each link has a static weight (W), similar to WSRR. The WD scheduler dynamically estimates the number of bytes (B) already scheduled for transmission in each link and schedules the next packet over the link with the lowest B/W ratio.

These modifications allow us to experiment with multiple 1- and 10-GBit/s Ethernet links and provide detailed measurements about performance and associated overheads.

3. Experimental Setup

Our experimental setup consists of two systems connected with multiple network interfaces. Both nodes have two Opteron 244 CPUs running at 1.8 GHz and a Tyan S2892 motherboard with 2 GBytes of main memory. The operating system is the 64-bit version of Debian with Linux kernel version 2.6.18.8 compiled with GCC version 4.1.2. Each node is equipped with the following network interfaces:

- One Myricom 10G-PCIE-8A-C card;

- One Intel PRO/1000 PT Quad Port PCI-E card;

- One Intel PRO/1000 GT Quad Port PCI-X card.

All links on both nodes are connected pairwise with cross-cables without a switch. In our experiments we vary the number of 1 GBit/s links from one to eight and we contrast our multiple-link results with the single Myrinet 10 GBit/s link.

Small MTU sizes decrease the ratio of payload data per packet, which limits the system's throughput. For this reason we believe that large MTU sizes are more representative of future trends. Thus, in our experiments we present results only for 9000-byte frames.

We evaluate the system using three micro-benchmarks:

- one-way: One of the two nodes sends messages back to back using remote writes without waiting for any response from the receiver, which passively receives data. This benchmark exercises the send path at the sending and the receive path on the receiving node.

- two-way: Both nodes simultaneously transmit data using back to back remote writes. The throughput in this case accounts for all data sent and received in each node, as both the send and receive paths are exercised at the same time.

- ping-pong: This is a request-reply benchmark using remote memory writes. Both request and reply are of the same size.

To understand the implications of using multiple links we use the following metrics:

- Throughput: the amount of useful payload data that has been delivered to the remote node. We present one curve for each link configuration. We use nxk to denote n links of k Gbits/s each.

- CPU utilization breakdown: the CPU time used for network processing. To represent the two CPUs available in our nodes, we use a maximum CPU utilization
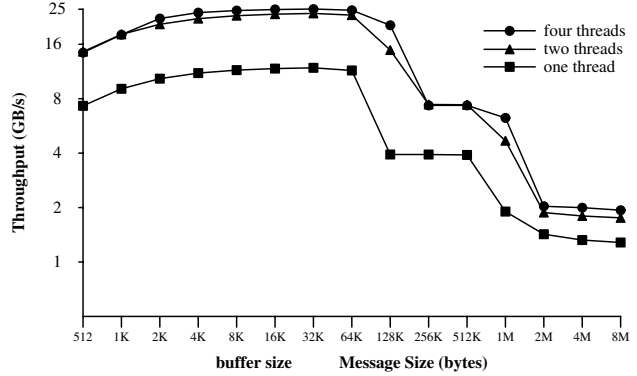

Figure 1. Memory copy throughput for a single node.

of 200%. Our CPU utilization results are approximate as we cannot account for the time between a NIC issues an interrupt, until the interrupt handler is executed on the host CPU.

For the breakdowns we use the following labels:

- Poll/IRQ: Interrupt handling or polling for servicing links.

- TxCopy: Copy data from user to kernel space in the send path.

- RxCopy: Copy data from kernel to user space in the receive path.

- Packet: Protocol overhead for packet preparation and processing.

- Device: I/O operations for handling hardware devices.

In our result, we present one CPU utilization breakdown bar per link configuration for each message size. The leftmost bar refers to the 1x1 configuration and the two rightmost to the 8x1 and 1x10 configurations respectively. In one-way we show breakdowns for both the sender and receiver. In two-way and ping-pong we only present breakdowns for one of the two communicating nodes; in these benchmarks both nodes behave in a similar manner because the send and receive paths of the protocol are exercised.

To better understand memory limitations in our system, we present briefly their memory configuration: The Opteron 244 CPUs in our system have a TLB size of 1024 entries and L1 and L2 cache sizes of 128 KBytes and 1 MByte respectively. Each processor is equipped with 4 DIMMs of 256 MBytes DDR-333 for a total of 2 GBytes main memory.

Figure 1 shows the memory throughput in the nodes we use. Sustained memory copy throughput for a single CPU
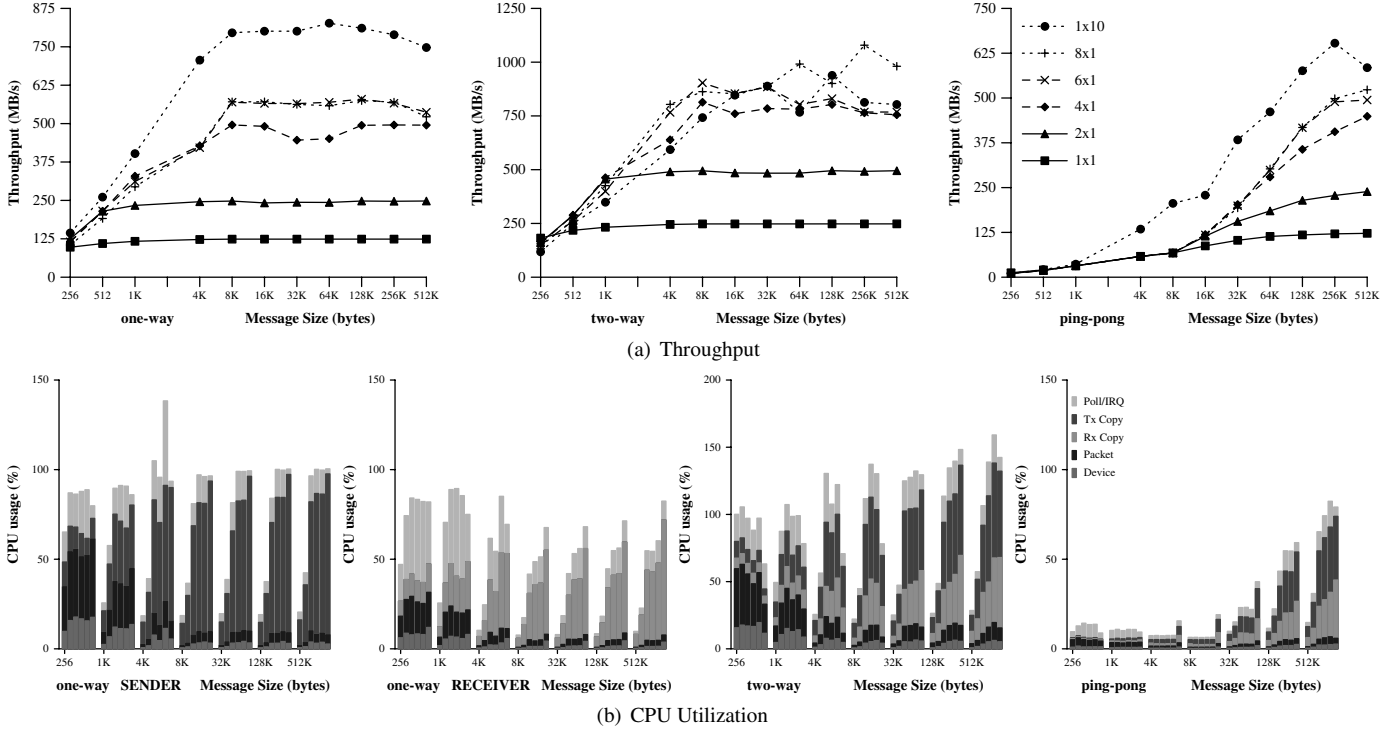
(a) Throughput

(b) CPU Utilization

Figure 2. Throughput and CPU utilization in the base system. For CPU utilization we show one bar per configuration, with the leftmost bar referring to 1x1 and the two rightmost bars to 8x1 and 1x10 respectively.

is about 1.3 GBytes/s. For copy sizes up to 64 and 512 KBytes, copy throughput is higher due to L1 and L2 hits respectively. With two and four threads we see that memory throughput is doubled, as both CPUs transfer data to independent memory modules. Since a memory copy involves two data crossings over the memory bus, peak memory throughput is about 2.5 GBytes/s. Finally, Linux is configured with NUMA features enabled.

We do not present latency results, as in most configurations 4-byte messages exhibit similar one-way, end-to-end latency, between 18 and 22 μs. Finally, we organize our experiments around the following system configurations:

- *Base:* Our base protocol.

- *Polling:* Our base protocol with interrupts disabled and replaced by a polling mechanism.

- *noCopy:* Our base protocol with copies to and from user-space artificially disabled. In this configuration the exact amount of data is transferred over the network.

4. Results

4.1. Multiple-link overhead

Figure 2(a) shows data throughput for the base configuration. First, we see that in one-way and ping-pong benchmarks, data throughput is approximately proportional to the number of links. The throughput continues to increase for up-to six links but remains almost at the same level (about 550 MBytes/s) for eight links. On the other hand, the 1x10 configuration scales up-to 800 MBytes/s in the one-way benchmark. In the two-way benchmark, we see that throughput scales well up-to four links. With higher number of links, throughput fluctuates around 800-900 MBytes/s and exhibits a large variance (graphs are averages of three runs).

Second, throughput is limited be the available CPU resources. Figure 2(b) shows our CPU utilization breakdowns. One-way throughput is limited by the send path overhead. Our protocol is able to use a single CPU for send path execution, only partially offloading transfer completions to the second CPU. Two-way is limited by load imbalances in the protocol, which is not able to fully utilize both CPUs.
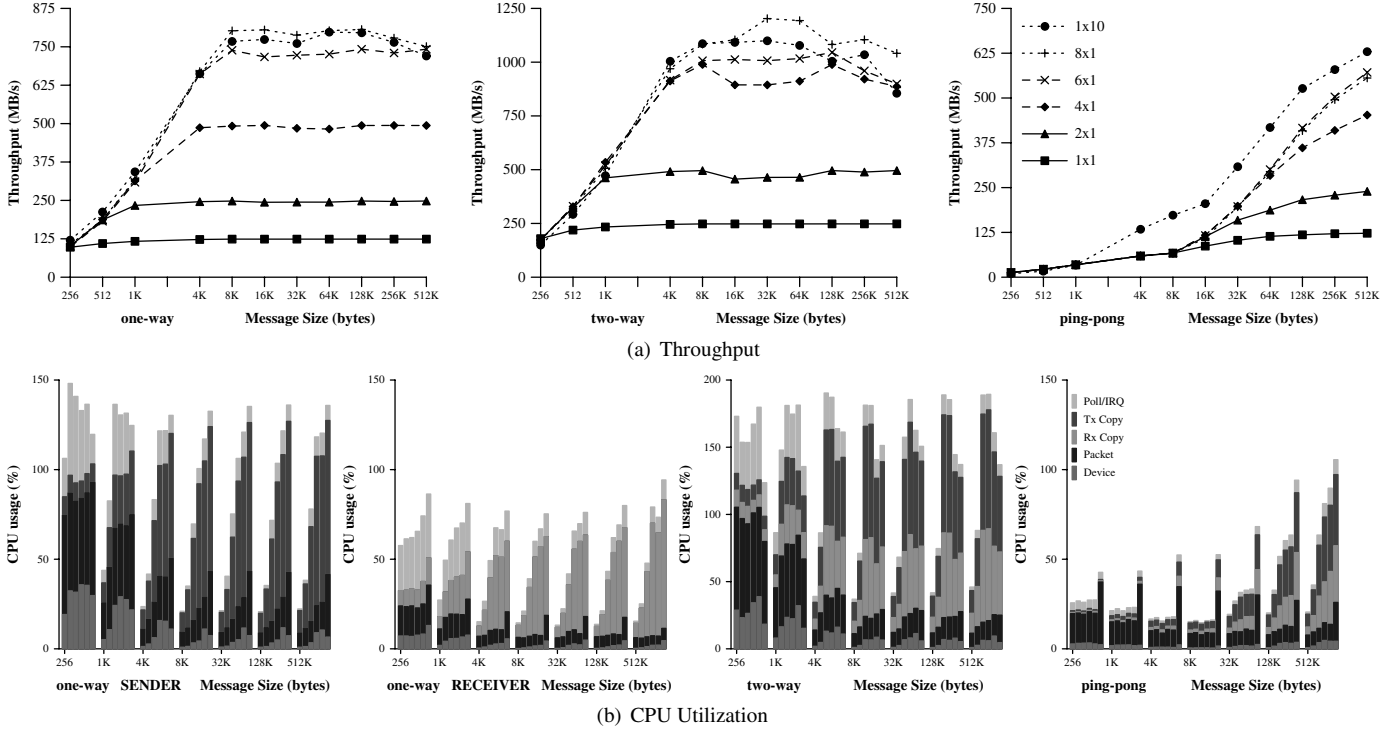
(a) Throughput

(b) CPU Utilization

Figure 3. Throughput and CPU utilization when interrupts are replaced by polling.

For all benchmarks we see that data copies are the dominant portion of the CPU utilization. Moreover, copy overhead increases proportional with message size. For example, with 128-Kbyte messages and 8x1 links, copy overhead accounts for 75% and 65% on the send and receive paths respectively in one-way, and about 70% on both send and receive paths in two-way and ping-pong.

Third, interrupt cost increases with the number of links but not the speed of each link. Note that for 4-KByte messages and 1x8 links, the interrupt overhead is higher than in other configurations as the hardware and protocol interrupt coalescing mechanisms are not effective, resulting in a large number of interrupts being delivered. Overall, the cost for interrupts, packet processing, and device management decrease when we increase the message size and remain at the same level for messages larger than 8-KBytes. Moreover, an increased number of dropped packets could potentially result in increased packet processing time. However, the number of dropped packets in all our experiments is negligible.

Finally, comparing 1x10 and 8x1 links, we see that 1x10 exhibits higher throughput in cases where the CPU utilization is lower, namely one-way and ping-pong. However, in two-way different message sizes result in different behavior.

4.2. Impact of interrupts

Figure 3 shows the data throughput and CPU utilization when interrupts are replaced by a polling mechanism.

Compared to Base, we notice that throughput increases up-to 38% and 25% in one-way and two-way respectively, but remains at the same level in ping-pong. This is due to two reasons: a) we avoid high interrupt pressure when there is large numbers (hundreds) of outstanding data transfers and b) the receive path is able to run on a different CPU from the send path. Between 1x10 and 8x1 links, we see that they have almost the same performance in one-way and two-way benchmarks. We believe that this is due to memory throughput limitations.

Moreover, similar to Base, one-way throughput is limited by send path processing. In two-way, 8x1 uses up both CPUs, whereas 1x10 is limited by send path processing, which is not offloaded to the free cycles of the second CPU.

4.3. Impact of copies

Figure 4 shows the data throughput and CPU utilization when copies are artificially disabled. We see that the one-way and ping-pong benchmarks exhibit throughput approximately proportional to the number of links, reaching the maximum possible throughput for each configuration. In comparison to Base with 1x8 links, throughput increases by
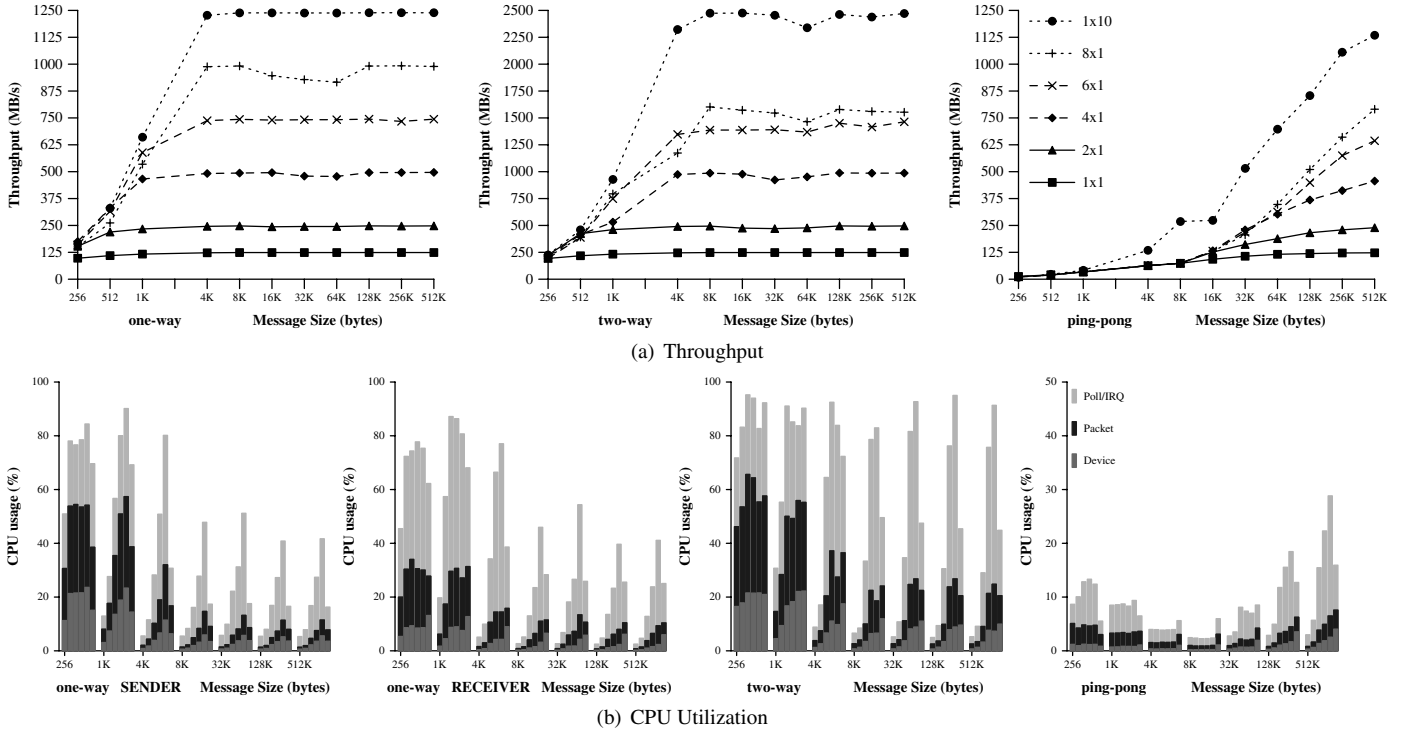
(a) Throughput

(b) CPU Utilization

Figure 4. Throughput and CPU utilization when copies are artificially disabled.

66,5% and 87,5% for one-way and two-way benchmarks respectively.

In all cases, a large portion of CPU time is spent servicing interrupts, and it increases proportionally with the number of links. This is also the reason that 1x8 configuration doesn't scale in two-way test.

4.4. Impact of scheduling

To examine the behavior of different schedulers, we present results for one configuration with *five* links, 1x10 and 4x1. Moreover, to avoid the limitations discussed above we disable both copies *and* interrupts. Figure 5 shows the data throughput and CPU utilization for the send path using different link schedulers.

We note that SRR treats all links in a similar manner and does not take advantage of the full link throughput available. WSRR and WD exhibit almost the same behavior in ping-pong, as the synchronous nature of the benchmark allows for only a small number of outstanding packets.

In one-way and two-way benchmarks, we see that for large message sizes, WD outperforms WSRR by 11% and 12.5% respectively. Finally, Figure 5(b) shows that for large messages WD spends more CPU time for packet management, which is affected by the time to prepare and schedule each data frame. The difference is approximately 10% and

20% in one-way and two-way respectively, similar to the throughput benefit of WD over WSRR.

5. Related Work

The last two decades there has been extensive research on communication subsystems for building cost-effective, high-performance clusters. To a large extend this research has focused on examining issues in the host-network interface, such as eliminating data copies, system call overhead in the communication path, and context switches [3, 9, 15, 19]. Through this work, network interface architectures have evolved dramatically to low-latency, high-throughput designs that are decoupled from the processor-memory architecture [1, 4, 5, 11, 17].

Similarly there has been extensive work in evaluating various aspects of cluster interconnects and in different contexts [2, 13]. Our work in this paper differs in that we examine extensively the impact of a large number of links on overheads related to the host-network boundary and we reveal related issues. Moreover, we do so using Gigabit Ethernet, in anticipation of its increased role in cluster interconnects and we contrast results with a state-of-the-art 10 Gbit/s setup.

In [12], *MultiEdge* has been evaluated on a cluster of 32 nodes, using single and dual Gigabit Ethernet links and sin-

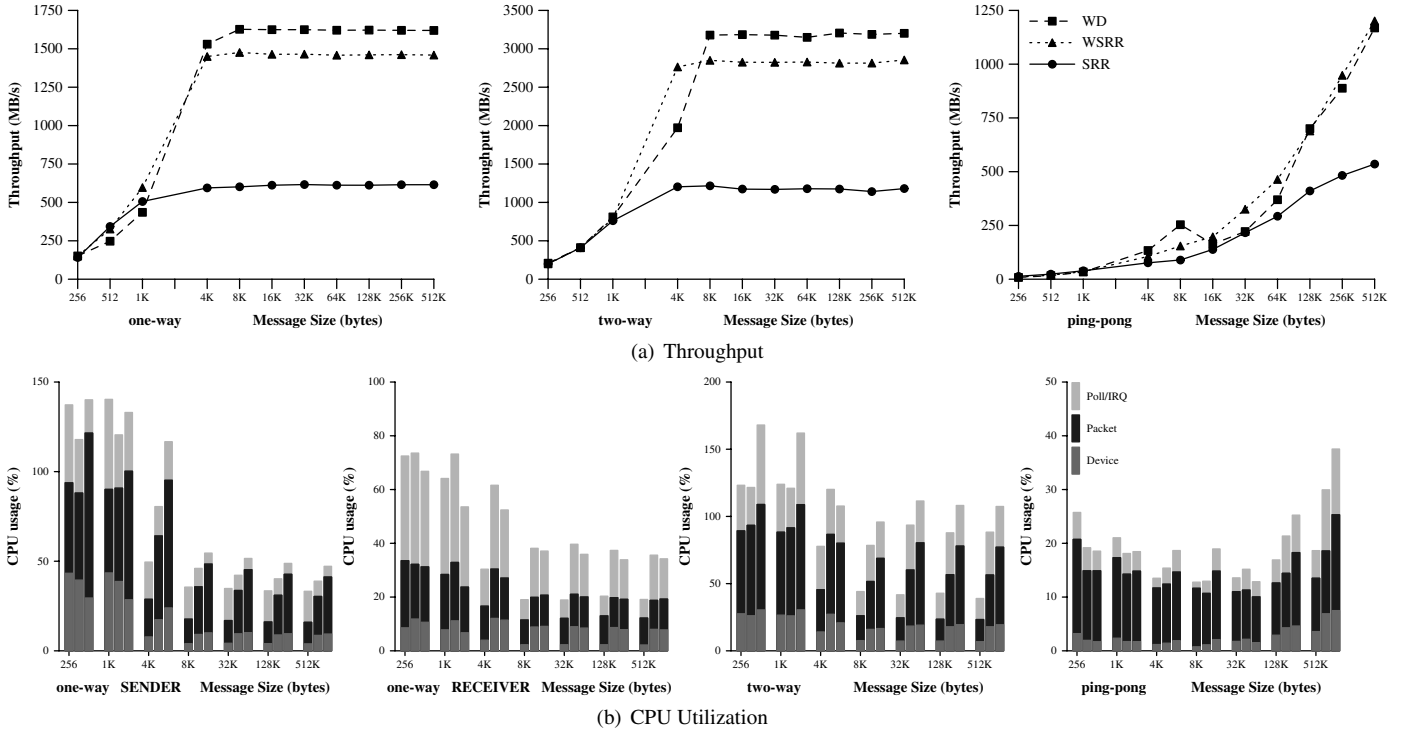(a) Throughput

(b) CPU Utilization

Figure 5. Throughput and CPU utilization when using different schedulers. In CPU utilization we show one bar for each scheduler: SRR (left), WSRR (middle), WD (right).

gle 10 Gbit/s Ethernet links, running an optimized software shared memory protocol and real applications. The results in [12] show that *MultiEdge* is able to deliver high throughput and more importantly, that the lack of protocol support from the network switches does not result in excessive extra network traffic for the system size examined. The difference in the throughput results between the base protocols in the two papers is primarily because of differences in the platform setup and secondarily due to changes in the protocol code.

The concept of end-to-end multi-link communication channels is similar to *inverse multiplexing* [10]. Inverse multiplexing has previously been applied to wide area network communication [7]. We use a similar concept in this paper but the tradeoffs and mechanisms required for our target platform, i.e., scalable systems, are different.

Moreover, this concept has been explored in the context of cluster interconnects: Multi-rail communication tries to take advantage of spatial parallelism and has been examined by previous work. The authors in [8] examine rail allocation methods for multi-stage cluster interconnects. They conclude that some allocation methods can yield significant improvements in latency and bandwidth. In our work we examine similar issues, however (a) using edge-based protocols and (b) by exploring a higher degree of spatial paral-

lelism. The authors in [14] examine the impact of multi-rail protocols on MPI applications by modifying higher system layers. In this work we focus to examine the impact of a larger number of physical links.

Finally, there are recent efforts to build multi-stage interconnects out of Gigabit Ethernet switches and network interfaces. The authors in [18] build a multi-dimensional hyper crossbar network using multiple Gigabit Ethernet interfaces in each node. They find that for a set of micro-benchmarks the system delivers more than 90% of the peak throughput. This work is orthogonal to our work in this paper as it focuses on the impact of the multi-stage interconnect rather than the degree of spatial parallelism.

The issue of load balancing protocol processing on multiple CPUs has been examined in [21]. The authors show the hardware extensions and synchronization required for distributed Ethernet processing on multiple CPUs in a high-speed network interface. In our work, we examine the limitations of using multiple host CPUs. We believe that our approach is inline with current technology trends of using multicore CPUs as host processors.

6. Conclusions

In this work we examine the implications on throughput and CPU utilization of using multiple 1-GBit/s links. We also contrast these configurations with a single 10-GBit/s link. We find that multiple link configurations are limited by a larger number of interrupts and poor load balancing of the send and receive paths over multiple CPUs. The impact of memory copies is significant, and removing them results in up-to 66% improvement in maximum throughput. Furthermore removing interrupts results in up-to 38% improvement in maximum throughput. We should also try to use more than one single CPU for each path, to utilize better the total CPU resources.

Overall, our results show that protocols for achieving high throughput over low-cost, commodity interconnects, using multiple links are a promising direction. However, future work needs to address issues related to (a) efficient coalescing or elimination of interrupts however, still supporting asynchronous modes of communication, (b) load balancing of protocol overheads to multiple CPUs without introducing excessive synchronization for protocol data structures, and (c) elimination of copies by avoiding NIC-level virtual to physical translation for cost and complexity reasons.

7. Acknowledgments

References

[1] An infiniband technology overview. Infiniband Trade Association, http://www.infinibandta.org/ibta.

[2] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin. User-space communication: A quantitative study. In *SC98: High Performance Networking and Computing*, Nov. 1998.

[3] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. *Proc. of the Fifteenth Symposium on Operating Systems Principles (SOSP15)*, December 1995.

[4] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. A virtual memory mapped network interface for the shrimp multicomputer. In *Proc. of the 21st International Symposium on Computer Architecture (ISCA21)*, pages 142–153, Apr. 1994.

[5] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovicm, and W. Su. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, 1995.

[6] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[7] F. M. Chiussi, D. A. Khotimsky, and S. Krishnan. Generalized inverse multiplexing of switched atm connections. In *Global Telecommunications Conference 1998 (GLOBECOM'98)*, volume 5, pages 3134–3140, Nov. 1998.

[8] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits. Using Multirail Networks in High-Performance Clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8):625–651, 2003.

[9] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. Vmmc-2: Efficient support for reliable, connection-oriented communication. In *Hot Interconnects V*, 1997.

[10] J. Duncanson. Inverse multiplexing. In *IEEE Communications Magazine*, pages 34–41, Apr. 1994.

[11] Giganet. Giganet cLAN family of products. http://www.emulex.com/products.html, 2001.

[12] S. Karlsson, S. Passas, G. Kotsis, and A. Bilas. Multiedge: An edge-based communication subsystem for scalable commodity servers. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, Mar. 2007.

[13] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. P. Kini, D. K. Panda, and P. Wyckoff. Microbenchmark performance comparison of high-speed cluster interconnects. *IEEE Micro*, 24(1):42–51, 2004.

[14] J. Liu, A. Vishnu, and D. K. Panda. Building multirail infiniband clusters: Mpi-level design and performance evaluation. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 33, Washington, DC, USA, 2004. IEEE Computer Society.

[15] S. Pakin, V. Karamcheti, and A. Chien. Fast Messages (FM): efficient, portable communication for workstatin clusters and massively-parallel processors. *IEEE Concurrency*, 1997.

[16] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The quadrics network (qsnet): High-performance clustering technology. In *Proc. of The 2001 IEEE Symposium on High Performance Interconnects (HOT Interconnects 9). Stanford, CA, USA.*, Aug. 2001.

[17] F. Petrini, W. Feng, A. Hoisie, S. Coll, and F. E. The quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, 2002.

[18] S. Sumimoto, K. Ooe, K. Kumon, T. Boku, M. Sato, and A. Ukawa. A scalable communication layer for multidimensional hyper crossbar network using multiple gigabit ethernet. In *Proc. of the 20th ACM International Conference on Supercomputing (ICS06)*, pages 107–115, June 2006.

[19] H. Tezuka, A. Hori, and Y. Ishikawa. PM: a high-performance communication library for multi-user parallel environments. Technical Report TR-96015, Real World Computing Partnership, 1996.

[20] TOP500.Org. Top500 supercomputing sites, Nov. 2007. http://www.top500.org.

[21] P. Willmann, H. Kim, S. Rixner, and V. Pai. An efficient programmable 10 gigabit ethernet network interface card. In *International Symposium on High Performance Computer Architecture (HPCA), San Francisco, CA*, Feb. 2005.