Towards an Integrated IO and Clustering Solution using PCI Express

Venkata Krishnan

Dolphin Inc.

Marlborough, MA 01752

krishnan@dolphinics.com

***Abstract*— PCI Express (PCIe), the IO interconnect of choice in today's single host computing platform, is being enhanced to support features that include I/O virtualization and processor-coprocessor interconnect. Host-to-host communication, however, is regarded as beyond the scope of PCIe. As such, the model in today's multi-compute platforms is to utilize PCIe for communication between the host(s) and the IO subsystem while a dedicated clustering interconnect such as Infiniband or Ethernet is used for host-to-host communication.**

The Dolphin Express solution (i.e. Dolphin's enhanced PCIe hardware and accompanying software) addresses the shortcoming of host-to-host communication in PCIe in a cost-effective manner. To the best of our knowledge, these products provide an industry first solution of using a PCIe-based switch fabric to seamlessly integrate both IO and clustering capabilities ─ thereby obviating the need for an additional clustering interconnect.

As a first step towards providing a full-fledged clustering solution, we have enabled support for TCP/IP protocol over PCIe (IPoPCIe). Based on benchmarking results, IPoPCIe was able to achieve performance that was on par or better than that of a 10GigE NIC's. On a 2-node system connected using a switch, IPoPCIe shows an end-to-end application latency of ~14µs and bandwidth of up to 1270MB/s.

Though the results are promising, the performance of IPoPCIe is greatly influenced by overheads associated with the TCP/IP protocol. Hence, work is ongoing on supporting a sockets direct interface that bypasses the TCP/IP stack. This would utilize the full potential of the underlying Dolphin Express hardware and lead to further reduction in latency, lower CPU utilization and increased bandwidth.

I. INTRODUCTION

PCI Express (PCIe) is fast becoming the de-facto IO fabric. With the level of support enjoyed by PCIe, it would not be an overstatement that PCIe would displace PCI in the foreseeable future. PCIe addresses many of the shortcomings of PCI and offers an order of magnitude higher performance than what PCI can offer ─ the current specification[1] doubles the lane bandwidth and endows PCIe with a bandwidth potential of up to 16Gb/s. Nevertheless, the single most important reason behind PCIe's success may not lie in its technical strengths but rather in its innate ability to support legacy PCI software. The fact that PCIe was designed to be completely transparent to PCI-software has been, without a doubt, instrumental in its wide-scale adoption.

PCIe has also continued to evolve beyond the traditional PCI realm wherein a single host manages a set of IO devices. PCIe will shortly support multi-host IO disaggregation and virtualization[2] and may also be used as a processor-coprocessor interconnect[3]. Going a step further, one may consider utilizing PCIe as a full-fledged clustering interconnect. However, to provide this level of functionality, PCIe should be able to prevent unfettered access to a memory region of one host by another host and in addition, support a messaging paradigm. This calls for PCIe to be augmented along the lines of Ethernet or Infiniband[4].

In the software world of cluster communication, the sockets API or its variants have been the de-facto interface for a wide body of networking applications that include web servers, content management systems, application servers etc. The level of entrenchment of sockets-based software is comparable to that of PCI-based software. As a result, most interconnect technologies[4][5][6] attempt to provide a completely transparent socket programming model[7][8][9]. This allows legacy applications to run without any code modification or recompilation. At the same time, the applications can utilize the full potential of the underlying interconnect.

Hence, for PCIe to be used as a clustering interconnect, the requirements are rather stringent – PCIe must be enhanced to include advanced data movement capabilities. The enhanced PCIe fabric must support legacy networking applications. And finally, the enhanced PCIe fabric must be able to seamlessly accommodate PCIe IO devices in a multi-host environment without resorting to PCIe bridging devices.

Dolphin's PCIe products, referred to as the Dolphin Express solution, satisfy the above requirements. This solution permits the use of PCIe for clustering while permitting host-to-PCIe IO device communication within the same fabric. To the best of our knowledge, this is the industry first solution for integrating IO and clustering onto a single PCIe based interconnect – a solution that obviates the need for a dedicated cluster interconnect.

Needless to say, Dolphin Express' solution will support legacy clustering APIs such as sockets, MPI, uDAPL. And in addition, the solution will also support Reliable Datagram sockets (RDS)[10] used in database clustering[11]. As a first step towards a comprehensive PCIe based clustering software solution, support for TCP/IP/sockets has been put in place.

This paper gives an overview of the Dolphin Express hardware but primarily focuses on TCP/IP over PCIe (IPoPCIe) support. The software infrastructure is described along with detailed evaluations that show that IPoPCIe is able to achieve performance that is on par or better than that of a 10GigE NIC's performance.

The rest of the paper is organized as follows: Section II provides some background and the motivation for integrating IO and clustering on PCIe. Section III gives an overview of the Dolphin Express hardware components; Section IV describes the IPoPCIe infrastructure; Section V evaluates IPoPCIe and compares it with a 10GigE NIC; finally Section VI discusses future work and concludes the paper.

II. Background

IO disaggregation, which involves isolating the IO subsystem from the main processor complex, provides immense benefits on several fronts. It enables high availability – when a primary IO device fails, a redundant IO device can take over without the need for bringing down the processor sub-system. It provides for efficient utilization of resources – IO resources can be apportioned to different processor sub-systems according to needs that vary over time. It enables cost-effective upgrades – the IO and processor subsystems can be treated as separate entities and hence, can be upgraded independently.

In a nutshell, IO disaggregation enables a fully virtualized IO model in which multiple processor subsystems can dynamically share a given set of IO resources. Currently, PCIe provides a naive IO disaggregation model wherein each host must have a dedicated PCIe fabric with its own PCIe switch(es) and IO devices. Any device reassignment to another host requires the IO device to be physically relocated from one host's PCIe fabric to the other host's.

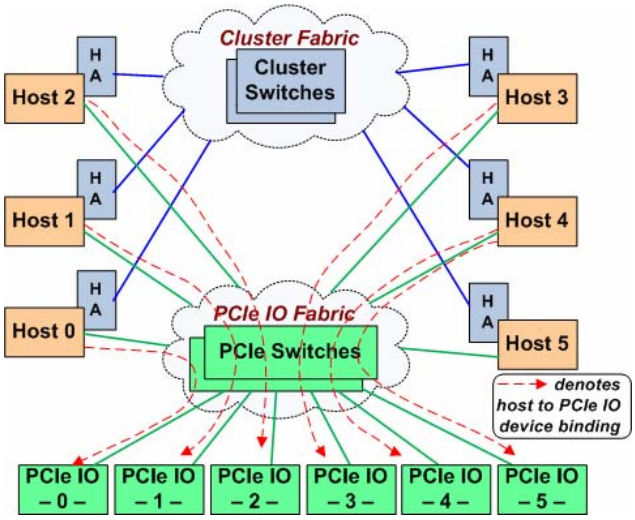


Figure 1 A dual interconnect model for IO and clustering needs.

PCIe is continuing to evolve and future PCIe products will address the above mentioned problem. Soon, PCIe will support complete IO virtualization (IOV) in a single root (SR) and multi root (MR) environment. The SR-IOV model is applicable only in a single host system. It focuses on providing functionality to an IO device such that it can be efficiently utilized by multiple system images on a single host (root). MR-IOV extends the SR-IOV concept to a multi-host platform. Hence, for supporting multiple hosts as well as providing flexibility in IO device assignment to an arbitrary host, MR-IOV capability is required in the PCIe switch fabric. And, if the IO device is also MR-IOV capable, a single device can also be shared across multiple hosts.

Nonetheless, there are no enhancements under consideration to enable PCIe for host-to-host communication. Consequently, the paradigm for building a multi-host system platform would involve PCIe for the host-to-IO communication component and a clustering interconnect such as Infiniband or Ethernet for providing host-to-host communication.

Figure 1 illustrates the dual interconnect model for supporting IO and clustering. The clustering interconnect is comprised of cluster switches and host adapters while the PCIe interconnect, as shown in the figure, comprises of MR-IOV capable PCIe switches. In this illustration, the host systems provide native PCIe interfaces and hence, no additional bridges or adapters are required to interface to the PCIe IO fabric. This is in contrast to the cluster fabric wherein a host adapter (HA) or a network interface card (NIC) is required on the host side for connecting to the cluster. While the cluster fabric enables host-to-host communication, the PCIe IO fabric permits a flexible IO disaggregation model and allows one or more IO devices to be bound to a host in a transparent manner.

Dolphin Express provides a clustering solution using an enhanced PCIe interconnect thus obviating the need for a dedicated clustering interconnect. To the best of our knowledge, this is an industry-first solution in this regard. The products enhance PCIe for clustering while permitting IO disaggregation using the same fabric. The following section details the Dolphin Express hardware components.

III. Dolphin Express Hardware

Dolphin Express hardware products provide robust host-to-host and native PCI Express host-to-IO functionality using a single fabric. The integrated PCIe clustering and IO fabric (henceforth referred to as PCIe* fabric) consists of three Dolphin Express hardware components (a) host adapter (HA) (b) cluster switch and (c) cluster/IO expansion switch.

The HA supports data movement protocols that permit host-to-host and host-to-IO communication. The host-to-host data movement protocol has two modes of transfer. The first mode of transfer uses a *send/receive* model wherein messages are transmitted to a queue at the receiving host system. The HA supports multiple such receive queues – these queues can be used to support a connection-less datagram model. The HA supports additional functionality by limiting queue access to a (i) single host (ii) set of hosts (iii) particular process in a specific host/set of hosts. The second mode of data transfer is memory based – the HA supports RDMA mechanisms for accessing the system memory of a remote host. The need for

exposing the physical memory for remote access is avoided by the use of security tags and more importantly, a memory indirection mechanism. Thus, any remote memory access is screened and validated by the HA before being directed to system memory.

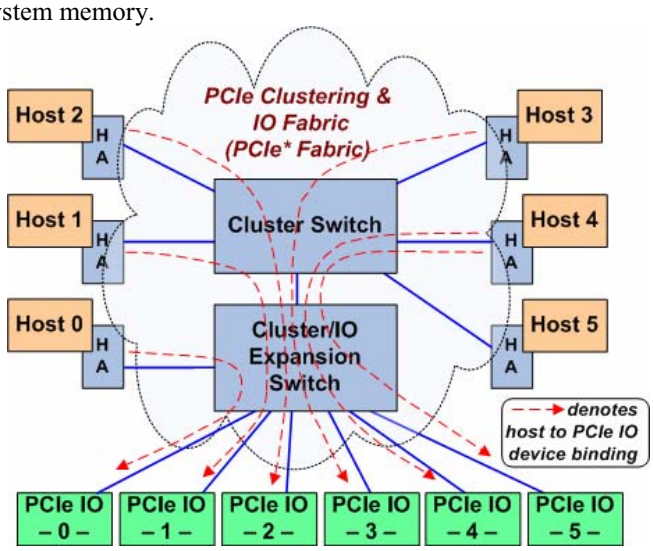


Figure 2 A single (enhanced) PCIe interconnect for IO and clustering needs.

The HA is connected to either a cluster switch or a cluster/IO switch in a PCIe* fabric. The interface to the fabric can be configured as one or two ports - either x8 (16Gb/s) or dual x4 (8Gb/s) PCIe lanes. The HA interface to the host is via x8 PCIe lanes (16Gb/s bandwidth).

The cluster switch provides connectivity between multiple hosts and/or the cluster/IO expansion switch. Finally, the cluster/IO expansion switch provides direct connectivity between hosts and PCIe IO devices. Note that the cluster/IO expansion switch interfaces directly to the PCIe IO device without requiring any additional bridge. The cluster/IO expansion switch functions as a cluster switch and provides a limited level of PCIe MR-IOV functionality. It is limited in the sense that it can support multiple PCIe hierarchies within a single interconnect fabric but cannot support sharing of a MR-IOV device across multiple hosts. For small clusters with a few IO devices, a cluster/IO switch may be sufficient. Alternatively, a cluster/IO expansion switch is not required for a configuration that doesn't entail IO disaggregation. By and large, the number of cluster switches and cluster/IO expansion switches will depend on the number of hosts and PCIe IO devices in the PCIe* fabric.

Figure 2 illustrates the usage of the HA, cluster switch and cluster/IO Expansion switch. In this setup, host 0 is connected directly to the cluster/IO expansion switch while hosts 1 through 5 are connected to the cluster switch. Furthermore, each host (except host 5) is also "bound" to one or more PCIe IO devices. For instance, host 1 is bound to PCIe IO device 1 while host 4 is bound to PCIe IO devices 4 and 5. Overall, this configuration enables direct communication not only amongst the six hosts but also between a host and its assigned PCIe IO device(s). The configuration also allows PCIe IO devices to be reassigned (via software) from one host to another host.

Though PCIe IO devices are shown in Figure 2, the cluster/IO expansion switch can natively support upcoming SR-IOV capable PCIe IO devices or PCIe switches. The virtualization features offered by the SR-IOV device is made available to the host with which it is bound to (via the PCIe* fabric). For now, the cluster/IO expansion switch will defer support for MR-IOV capable PCIe devices – devices that can be simultaneously shared across multiple hosts. Such devices will be treated as SR-IOV devices.

Instead of a PCIe IO device, a PCIe switch can also be used to interface to the cluster/IO expansion switch. All PCIe devices downstream of the attached PCIe switch are bound to a single host. In Figure 2, a PCIe switch can be attached to the cluster/IO Expansion switch and dedicated to host 4. If one were to assume that the underlying PCIe switch supported four PCIe IO devices, then host 4 would have access to a total of six PCIe IO devices (two IO devices directly via the cluster/IO Expansion switch ports and four devices via the PCIe switch).

IV. IPoPCIe

TCP/IP communication across the PCIe fabric is achieved by replacing the Ethernet driver with the IPoPCIe driver. Figure 3 illustrates the Dolphin Express software stack. Note that the Ethernet driver is shown for illustrative purposes only.

To reduce the CPU utilization on the *send* side, most Ethernet NICs support the TCP segment offload feature (TSO). In this mode, the NIC advertises a large MTU size (typically 64KB) to the TCP/IP layer. The NIC assumes the responsibility of fragmenting a large IP datagram into appropriate smaller sized IP datagram fragments so as to fit within standard size 1500-byte Ethernet frames (or jumbo sized 9000-byte frames). On the receive side, it is the responsibility of the CPU to reassemble the IP fragments into a large IP datagram, which is then passed on to the upper layers of the TCP/IP protocol.

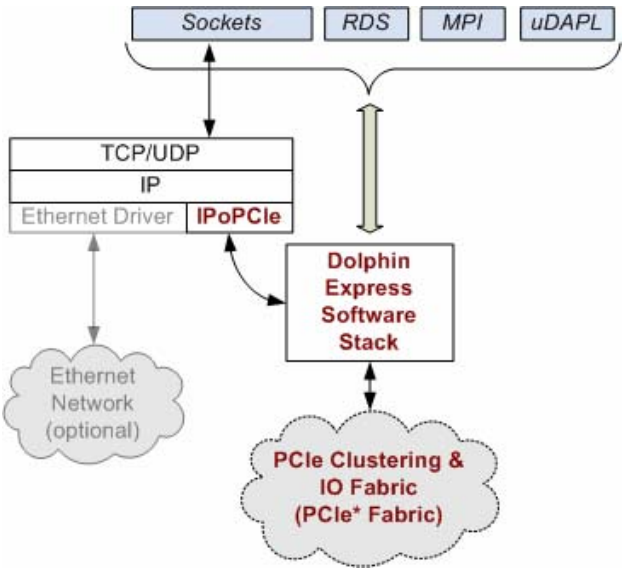


Figure 3 Dolphin Express software stack.

The IPoPCIe driver supports an enhanced TSO feature to the TCP/IP layer that, in addition to reducing the CPU utilization on the *send* side, also does the same on the *receive* side. By exploiting the RDMA features offered by the HA, a maximum sized IP datagram (64KB) is transmitted in toto and not subject to IP fragmentation. Of course, the large packet is fragmented at the PCIe* level into small packets (256 bytes) but it is the receive side HA, rather than software, that is responsible for reassembling the small packets into a large IP datagram. Details on the data transfer mechanism follow.

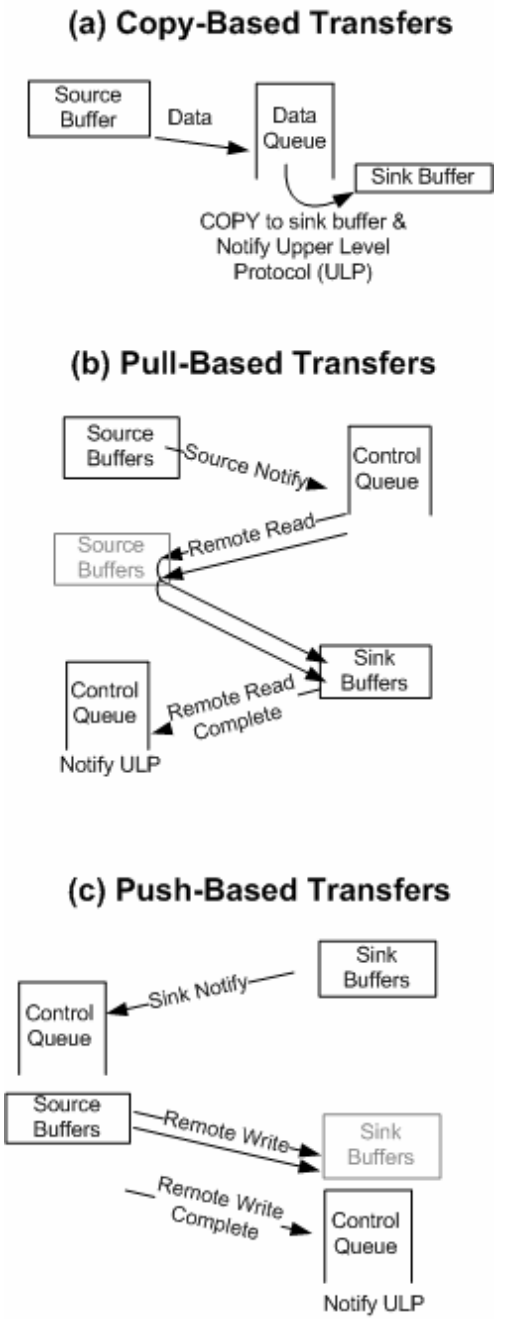


Figure 4 An outline of the underlying connection-less data transfer protocol used by IPoPCIe. The protocol incorporates three mechanisms for data transfer (a) Copy based (b) Pull-based (c) Push-based.

The IPoPCIe driver is built upon the underlying Dolphin Express software stack. The Dolphin Express stack supports three modes of transfer: (i) Copy-based (ii) Pull-based (iii) Push-based. This is illustrated in Figure 4. The copy-based mode of transfer is used exclusively for transmitting small data packets (< 256 bytes) and for control messages. These messages utilize the receive queues provided by the HA. The Pull- and Push-based mode of transfers, are in essence RDMA read and RDMA write transactions, and are used for transmitting large IP datagram packets.

The copy-based mode of transfer transmits data from a source buffer to a remote host's queue. When the size of the IP packet exceeds a threshold, the pull-based mode of transfer is utilized. As shown in Figure 4(a), a source notification message is sent to a queue. This control message is processed by the Dolphin software stack on the sink node and results in the initiation of a RDMA read operation. On successful completion of the read operation, the source node is notified. On receipt of the completion, the driver notifies the upper layer to free the corresponding transmit buffer.

The pull-mode of transfer, as shown in Figure 4(b), eliminates the need for fragmenting a large IP datagram. A 64KB sized packet is read in its entirety from the source buffer and placed directly onto the sink buffer. The benefits of avoiding IP fragmentation notwithstanding, a significant overhead is associated with the pull-based transfer operation – the source notification and the read operation adds a round-trip delay. Based on the initial set of results, there indeed was significant scope for improving performance in terms of latency as well as bandwidth. The end result was to incorporate a push-based transfer as part of the data movement.

Figure 4(c) illustrates how the source node can be notified about the sink buffers apriori using a control message. The efficiency of the transfer is greatly improved in that it permits the placement of data directly into the sink buffers as soon as the source data is available. The transition between pull-based and push-based transfer is done in a seamless fashion. Regardless of the type of transfer used, all protocols utilize the HA's two DMA engines to transfer data directly from/to the fabric to/from the system memory.

V. Experimental Results

All experiments were done in a Linux environment and utilized the TCP/IP networking benchmark, Netperf[12], for measuring the network latency and bandwidth. The configuration consisted of two x86 hosts (3.6 GHz Intel Xeon based and 2GB RAM). A wide range of packet sizes were used for studying the latency and bandwidth.

A. Baseline

Towards establishing a baseline for comparison, the performance of TCP/IP over 10GigE was evaluated. The 10GigE cluster configuration used for the experiments were relatively straightforward – the two x86 hosts were connected point-to-point without any intermediary switch. The 10GigE NICs used for the evaluation were Myricom's Myri-10G PCI

Express based adapters[13] that supported a x8 PCIe interface to the host The NIC and TCP/IP parameters were tuned as suggested by the vendor to achieve the maximum performance.

1) Latency: An important feature supported by the 10GigE NIC was its ability to coalesce interrupts via a (configurable) timer setting. Minimizing the number of interrupts by using a large timer setting is indeed beneficial during large data transfers as it lowers CPU utilization. At the same time, the coalescing mechanism affects the latency in an adverse fashion. In an attempt to balance out the impact on bandwidth and latency, Myri-10G NIC uses a default timer of 25µs. For the experiments, in addition to this default value, two additional timer settings of 0µs and 50µs were used – the timer value of 0µs disables interrupt coalescing entirely thereby permitting the best latency numbers to be realized while the timer value of 50µs reduces CPU utilization and provides the best bandwidth numbers during large data transfers.

Furthermore, jumbo Ethernet frames (MTU = 9000) were enabled for the experiments. This resulted in lower latency when compared to using standard Ethernet frames (data not shown) particularly for large messages. For small messages, the latency was nearly identical. The use of jumbo Ethernet frames does not impact the latency of small messages in this experimental two-node configuration when using a well-behaved application like the *Netperf* benchmark. However, jumbo frames are typically avoided in real-life configurations with a wide range of packet sizes and particularly when running latency sensitive applications.

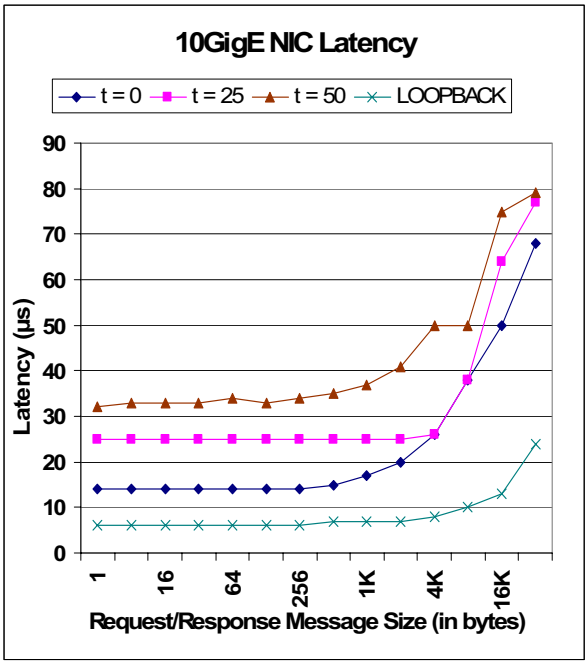


Figure 5 Illustration shows the latency for varying message sizes as well as the effect of changing the interrupt coalescing timer. A timer value of 0 (t = 0) indicates that interrupt coalescing is disabled.

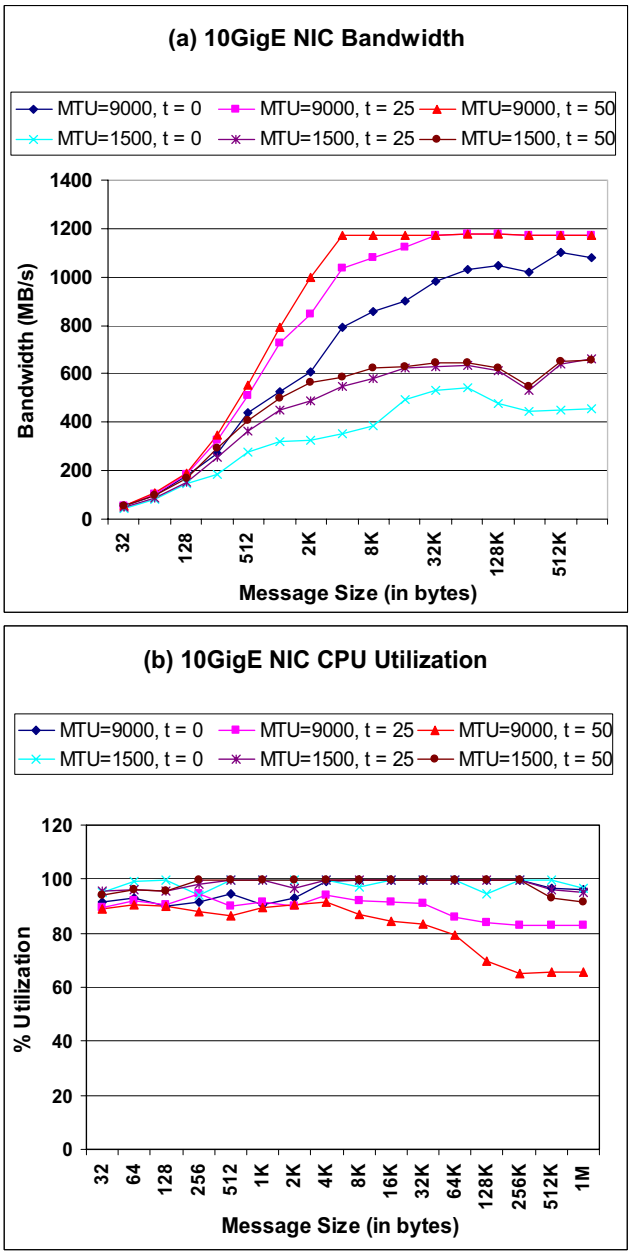


Figure 6 Bandwidth and receive side CPU utilization for a 10GigE NIC using jumbo frames (MTU = 9000) Chart (a) shows how bandwidth varies with (i) message sizes (ii) interrupt coalescing timers. Chart (b) shows the receive side CPU utilization. Both charts also include data when standard Ethernet frames (MTU = 1500) are used.

Figure 5 shows the average latency for different message sizes (ranging from 1-byte to 32-KB) and varying interrupt coalescing timer values (0µs, 25µs and 50µs). Predictably, the lowest latency (for a 1-byte request/response message) of 14µs is observed when the interrupt coalescing feature is completely disabled (t = 0). And as expected, the latency for a 1-byte transfer increases when the timer values are increased. The 1-byte packet latency increases to 25µs (when t = 25µs) and 32µs (when t = 50µs). For the sake of completion, the figure also includes the latencies obtained by loop-back. Since the same CPU was utilized for both send as well as receive

side TCP/IP stack processing in the case of loopback, the impact on latency for small packets is substantial. Nonetheless, the latency of 6µs provides a measure on the lower bound on latency that could be achieved when the TCP/IP stack is not bypassed.

2) *Bandwidth:* Unlike for latency, interrupt coalescing is indeed beneficial for improving bandwidth. Figure 6(a) shows the bandwidth while Figure 6(b) shows the corresponding receive side CPU utilization for different message size (32-bytes to 1MB) and interrupt coalescing timer values (0µs, 25µs and 50µs). It is obvious that there is a significant increase in bandwidth when using a large timer due to the reduced interrupt processing. For instance, in Figure 6(a), one can see that the bandwidth when MTU = 9000 and timer = 0µs always trails the bandwidth when MTU = 9000 and timer = 50µs. This is due to the increase in CPU utilization when the timer is disabled – see Figure 6(b).

As mentioned in an earlier section, the use of jumbo frames may not be appropriate in latency-sensitive environments. Hence, the experiments were run using standard sized Ethernet frames. Predictably, the use of standard Ethernet frames came at a cost. the additional CPU overhead on the receive side due to IP packet reassembly resulted in a significant bandwidth drop ─ from a maximum bandwidth of 1175MB/s (when using jumbo frames) to a maximum bandwidth of 660MB/s. Overall in this 2-node configuration, the highest bandwidth is observed when using jumbo frames and interrupt coalescing is enabled (with timer = 50µs).

Even assuming that jumbo frames were enabled, the NIC interrupt timer settings required to achieve the lowest latency conflicts with the settings required for achieving the highest bandwidth. A timer vale of 0µs gives the lowest latency of 14µs and a maximum bandwidth of 1100MB/s while a timer value of 50µs results in the highest bandwidth (1175MB/s) but at the cost of higher latency (32µs). Nonetheless, in the following section, only the best performance numbers from Figure 5 and Figure 6 are chosen for comparison with IPoPCIe.

B. IPoPCIe Performance

The two hosts used for the 10GigE evaluation were also used for the IPoPCIe study. For this 2-host configuration, a cluster switch was nevertheless included so as to satisfy a minimal cluster/IO configuration as illustrated in Figure 2. From a latency perspective, the switch has little impact (< 0.5µs) and hence, a direct comparison can be made with the point-to-point 10GigE configuration.

The HA supports an x8 PCIe interface to the host. On the PCIe* side, the HA can be configured as either x4 or x8 lanes. For the purpose of the study, two types of PCIe* connectivity were evaluated: one using x4 lanes and the other using x8 lanes. For x4 lanes, the effective bandwidth available is 8Gb/s whereas for x8 lanes, it is 16Gb/s. Taking into account the packet efficiency of ~85-90% (based on header and other link overheads) on the PCIe* interface when using maximum sized 256-byte packets, the realizable bandwidth is ~7.2Gb/s and ~14.4Gb/s for x4 and x8 lanes respectively.

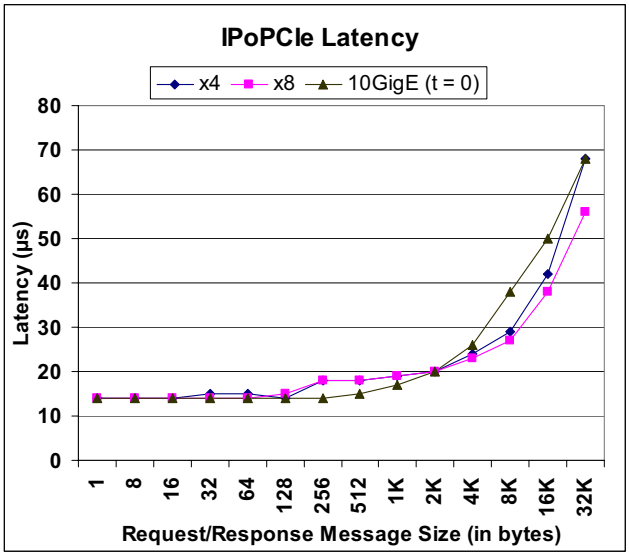


Figure 7 Comparing the IPoPCIe latencies with that of the best latencies obtained on a 10GigE setup (wherein interrupt coalescing is completely disabled)

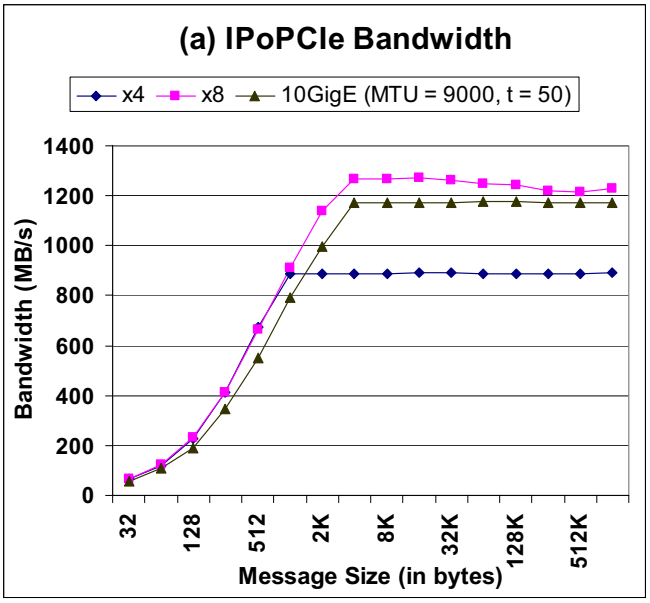
1) Latency:

Figure 7 charts out the latencies for IPoPCIe for the x4 and x8 configuration. The chart also includes the *best case* latencies observed for the 10GigE setup when interrupt coalescing was completely disabled. It can be observed that, for small packets (1 byte – 256 bytes), the IPoPCIe latencies of ~14µs (for both x4 and x8 configurations) matches that of the 10GigE NIC's latencies. For message sizes of 256 bytes and beyond, IPoPCIe transitions from a copy mode of transfer to a push- or pull-based transfer. The additional overhead caused by the control messages has an impact on the latency and as a result, the observed latencies between 256 bytes to 1KB are higher by 1 – 3µs than that of the 10GigE NIC's. For message sizes of 1KB and 2KB, the time taken for the actual data transfer dominates the latency and consequently the latencies of both IPoPCIe and 10GigE NIC are nearly the same.

Moving past 2KB sized messages, the additional bandwidth allows the x8 configuration to achieve a lower latency than that of the x4 configuration. Interestingly, even with a lesser bandwidth when compared to the 10GigE NIC, the x4 configuration latencies are better until the message sizes reach 32KB. This is due likely to the software overhead of reassembling IP fragments on the receive side – this overhead is obviated by the enhanced TSO feature offered by IPoPCIe (see Section IV). And finally, when compared to the x8 configuration, the 10GigE NIC latencies are higher by over 10µs for messages 8KB and beyond.

2) *Bandwidth:* Though the theoretical bandwidth of the PCIe x8 interface to the host is 16Gb/s, there are other factors in play that further bring down the utilizable bandwidth. These include per-packet overhead of ~20 bytes as well as overheads for the PCIe link transactions (ACK/NACK etc.) Given that the server chip set supports a maximum PCIe packet size of 128 bytes, the overall packet efficiency of the PCIe host

interface is between 80 – 85% (i.e. ~13Gb/s). This is in contrast to the 85 – 90% efficiency on the PCIe* interface that supports a maximum packet size of 256 bytes. In an x4 PCIe* configuration, the bandwidth bottleneck is on the PCIe* side rather than the x8 PCIe interface. The maximum achievable bandwidth for the x4 PCIe* configuration is ~7.2Gb/s or ~900MB/s. On the other hand, in an x8 PCIe* configuration, the bandwidth bottleneck shifts to the x8 PCIe interface. Hence, the maximum achievable bandwidth for an x8 PCIe* configuration is ~13Gb/s or ~1600MB/s.



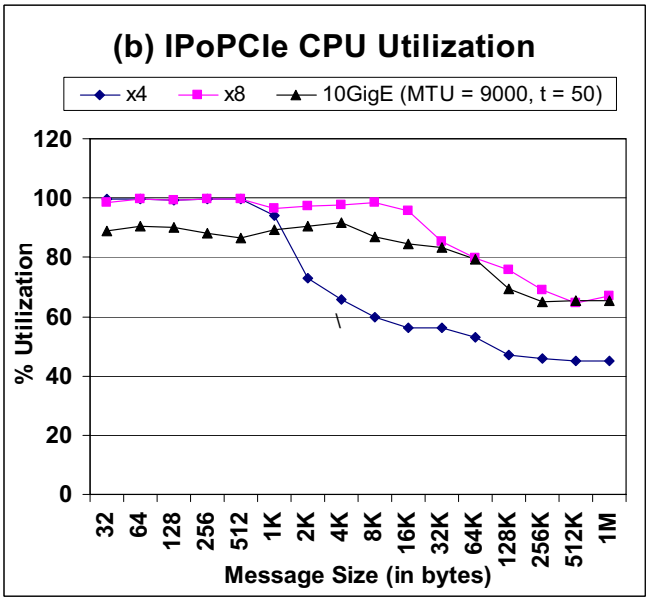


Figure 8 Comparing the IPoPCIe bandwidth and receive side CPU utilization with that of the best bandwidth/CPU utilization obtained on a 10GigE setup (interrupt coalescing timer set at 50µs and jumbo frames used)

Figure 8(a) shows the IPoPCIe bandwidth for the x4 and x8 configuration. The chart also includes the best case bandwidth for the 10GigE setup when the interrupt coalescing timer was set at 50µs. The bandwidth offered by both the x4 and x8 configuration is identical up to a message size of 1KB and exceeds that of the 10GigE NIC by over 100MB/s. Beyond 1KB message size, the bandwidth in the x8 configuration continues to improve and achieves a maximum bandwidth of ~1270MB/s while the x4 configuration saturates at the theoretical maximum bandwidth of ~900MB/s. In contrast, the 10GigE NIC achieves a maximum bandwidth of ~1170MB/s.

From the CPU utilization numbers in Figure 8(b), it is apparent that IPoPCIe suffers the overhead of the extra copy from the data queue to the sink (network) buffer. For larger message sizes, the push-mode or pull-mode transfer alleviates this problem and the impact of processing the control messages also disappears after 1KB for the x4 configuration. Beyond 1KB, the CPU utilization in the x4 configuration starts to drop and approaches ~45% for large message sizes.

The x8 configuration continues to have a 10% higher CPU utilization over 10GigE NIC – which roughly corresponds to the 10% extra bandwidth realized. For larger messages, the enhanced TSO feature (Section IV) permits the x8 configuration to offer the additional 10% bandwidth but with a CPU utilization that is on par or lower than that of the 10GigE NIC.

Interestingly, even though the CPU is not utilized completely, the x8 configuration's maximum bandwidth of ~1270MB/s is not close to the theoretical maximum of ~1600MB/s. This is in contrast to what was observed for the x4 configuration. Recall that TCP/IP protocol always involves a copy between the network buffer and application buffer. Hence, the processor needs to read/write the buffers from/into memory to perform the copy operation. Rather than the TCP/IP stack processing overhead, the effects of long memory latencies in fetching the buffers onto the on-chip cache start to play a dominant role when processing data at very high speeds. Indeed, any solution that does not place data directly onto the application buffers will invariably encounter the memory wall and would be unable to realize the full potential offered by the underlying interconnect. There have been attempts to solve the memory copy problem in a non-disruptive manner that would allow legacy TCP/IP applications to run without modifications – notable among them is the IO acceleration approach[14]. Dolphin's approach of addressing this memory access problem so as to achieve the full potential of the x8 interface is currently work in progress.

VI. Conclusions & Future Work

Dolphin's unique approach of enhancing the PCIe interconnect for host-to-host communication allows for the use of single interconnect for both IO and clustering needs – thereby eliminating the need for a dedicated clustering interconnect. The enhanced PCIe products incorporate capabilities to provide efficient yet secure host-to-host communication. In addition, the solution seamlessly accommodates existing PCIe devices without the need for additional bridges. Additionally, the IO component of the

solution provides complete flexibility in the (re)assignment of a PCIe IO device to an arbitrary host in a multi-host platform.

Dolphin Express solution will continue to support both legacy PCI/PCIe and clustering software. As a first step towards a comprehensive clustering software solution, support for TCP/IP over PCIe (IPoPCIe) has been enabled. Based on comparative results, IPoPCIe was able to achieve performance that was on par or better than that of a 10GigE NIC's. Though extremely promising, there is significant room for improvement given that the performance was affected by TCP/IP protocol overheads. Current work is focused on supporting Supersockets[15] - Dolphin's version of a direct sockets interface, which completely bypasses the TCP/IP stack. The use of this interface will lead to further performance improvement and endow the integrated IO/clustering PCIe* fabric the ability to be on par with dedicated clustering interconnects.

ACKNOWLEDGMENT

Many thanks to Lev Abidor, Randy Black, Lynne Brocco, Joe DeVincentis, Don Dutile, Ernie Grella, Dave Mayhew, Carey McMaster, Nitin Shah, Tom Tinory, Mike Vasicek, David Wong and Dan Woo for their valuable assistance during different phases of the project.

REFERENCES

[1] PCI Special Interest Group, "PCI Express 2.0 Base Specification," January 2007.

[2] PCI Special Interest Group, "I/O Virtualization Review Archive" *Restricted Access*

[3] Geneseo technology solution brief, Intel Corporation, November 2006. http://www.intel.com/technology/pciexpress/devnet/geneseotechnology.pdf

[4] Infiniband Trade Association, "Infiniband Architecture Specification, Release 1.2," Sep 2004. http://www.infinibandta.org

[5] A. Romanow. An Overview of RDMA over IP. In First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003), February 2003.

[6] The Dolphin SCI Interconnect, Dolphin Interconnect Solutions, Feb 1996. http://www.dolphinics.com/pdf/whitepapers/T-WhitePaper.pdf

[7] Infiniband Trade Association, "Sockets over Infiniband,", http://www.infinibandta.org

[8] Remote DMA (RDMA) Consortium, "Sockets Direct Protocol", http://www.rdmaconsortium.org/home/draft-pinkerton-iwarp-sdp-v1.0.pdf

[9] F. Seifert and H. Kohnmann, "SCI SOCKET - A Fast Socket Implementation over SCI," Dolphin Interconnect Solutions, Sep 2006. http://www.dolphinics.com/pdf/whitepapers/sci-socket.pdf

[10] R. Pandit, "Reliable Datagram Sockets (RDS)", OpenIB Developers Workshop, Feb 2006. http://www.openfabrics.org/archives/aug2005datacenter/Reliable_Datagram_Sockets.ppt

[11] P. Tsien, "RDS and Oracle 10g Update", InfiniBand Trade Association and OpenFabrics Alliance Joint Developers Conference, Sep 2006. http://www.openfabrics.org/archives/sep2006devcon/1530_oracle.ppt

[12] Netperf: A Network Performance Benchmark Revision 2.4.1, October 2005. http://www.netperf.org.

[13] Myri-10G PCI Express NIC with a 10GBase-CX4 port, http://www.myricom.com/Myri-10G/NIC/10G-PCIE-8A-C.html

[14] Greg J. Regnier, Srihari Makineni, Ramesh Illikkal, Ravi R. Iyer, Dave B. Minturn, Ram Huggahalli, Don Newell, Linda S. Cline, Annie Foong: TCP Onloading for Data Center Servers. IEEE Computer 37(11): 48-58, Nov. 2004.

[15] Dolphin Express & SuperSockets, Dolphin Interconnect Solutions. http://www.dolphinics.com/products/software/sci_sockets.html