

An Analysis of the Impact of MPI Overlap and Independent Progress

Ron Brightwell Keith D. Underwood
Scalable Computing Systems
Sandia National Laboratories *
PO Box 5800
Albuquerque, NM 87185-1110
{rbbright,kdunder}@sandia.gov

ABSTRACT

The overlap of computation and communication has long been considered to be a significant performance benefit for applications. Similarly, the ability of MPI to make independent progress (that is, to make progress on outstanding communication operations while not in the MPI library) is also believed to yield performance benefits. Using an intelligent network interface to offload the work required to support overlap and independent progress is thought to be an ideal solution, but the benefits of this approach have been poorly studied at the application level. This lack of analysis is complicated by the fact that most MPI implementations do not sufficiently support overlap or independent progress. Recent work has demonstrated a quantifiable advantage for an MPI implementation that uses offload to provide overlap and independent progress. This paper extends this previous work by further qualifying the source of the performance advantage (offload, overlap, or independent progress).

Categories and Subject Descriptors

D.4.4 [Communications Management]: Message sending

General Terms

Performance, Measurement

Keywords

MPI, message passing, overlap

1. INTRODUCTION

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000

The ability to efficiently overlap communication and computation can be a critical performance requirement for many applications. The emergence of OS-bypass communication technologies, like Myrinet [1], Quadrics [11], and InfiniBand [8], specifically address the desire to reduce the impact of message passing on the host processor by allowing the network interface adapter to perform communication functions asynchronously, thus achieving low host CPU overhead as well as low latency and high bandwidth. However, because few of these technologies provide adequate support for implementing MPI, the host processor overhead needed to support MPI semantics can potentially be significant.

Another area where most OS-bypass technologies fall short is in the way in which progress is made on outstanding non-blocking MPI communication requests. The MPI Standard mandates a Progress Rule for completion of asynchronous peer communication operations, but there is disagreement (even among the members of the MPI Forum) as to the exact semantics that this rule mandates. This disagreement has led to implementations that support making progress on outstanding non-blocking calls in different ways. Some implementations adhere to a strict interpretation, where outstanding operations make progress independent of subsequent calls to the MPI library. Others depend on the application to periodically make calls into the library so that outstanding requests can be progressed. This difference in how outstanding communication operations make progress can also have a significant affect on performance.

A third related characteristic of interest is the ability to offload MPI matching and protocol processing to an intelligent or programmable network interface. While all OS-bypass technologies support delivering data directly from the network into an application's address space, most networks require the host processor to perform MPI matching and queue traversal functions. Others, such as Quadrics, do not. Support for offloading MPI matching and protocol processing on the network interface rather than on the host can also significantly impact performance.

In this paper, we analyze overlap, progress, and offload in an attempt to characterize the effect that they have on the performance of an application. This analysis is performed on data gathered using several different MPI implementations that have nearly identical micro-benchmark performance, but that differ in their support of progress, overlap, and offload. We use application benchmarks to quantify and isolate the performance impact of these features and also identify application characteristics that influence their effectiveness.

The rest of this paper is organized as follows. In the following section, we provide more detail on the importance of overlap,

progress, and offload, as well as describe how this paper complements other published research. In Section 3, we describe the approach used for our analysis and discuss the platforms, MPI implementations, and application benchmarks that were evaluated. We continue in Section 4 with a presentation of our results, which are accompanied by a detailed analysis. Section 5 summarizes the important conclusions of this study, and Section 6 outlines possible avenues of further investigation.

2. BACKGROUND

This paper seeks to quantify the impact of overlap, independent progress, and offload from an application perspective. Rather than focusing simply on whether or how a network and its associated protocol stack can provide these features, we wish to quantify their benefit. The following discusses the importance of these features in detail, and outlines the contribution of this analysis relative to other approaches that have been taken.

Measuring the overlap potential of a network is straightforward, but measuring the impact of overlap on an application is not. Most micro-benchmarks that attempt to quantify overlap are written to measure the degree to which a network can satisfy the potential to completely overlap computation and communication. Few studies have been done that actually quantify how much overlap potential exists within an application. For applications that provide little opportunity to overlap, providing support for overlap may actually decrease performance.

Implementations of MPI that employ independent progress can potentially be more efficient than those that do not. This is particularly important for large messages, since they offer the greatest opportunity to overlap of computation and communication. Most implementations of MPI use a rendezvous protocol for exchanging a large message. This protocol involves the sender sending an initial message that describes the message to be sent. For networks that support RDMA read operations, the receiver can receive this request, use an RDMA read operation to pull the data directly from the sender's buffer, and then send a message to the sender to indicate that the transfer has been completed. For networks that do not support RDMA read operations, the receiver must send back information to the sender describing where the data is to be placed.

Suppose the receiving process posts a non-blocking receive before the send request arrives. When the send request eventually does arrive, all of the information needed to complete the request is available. For some network interfaces, like Quadrics, the RDMA read request (and subsequent completion message) can be issued without any further involvement from the receiving process or the host CPU. For implementations that depend on MPI calls to be made, the RDMA read operation cannot be issued until the application calls an MPI library routine. In this case, performance is dependent on how often MPI library calls are made when there are outstanding requests. This interval is independent of the network and is dependent on the structure of the application and on the operating system.

Overlap and progress are usually thought to be intertwined, but it is possible to have one and not the other. OS-bypass networks that do not provide independent progress are still able to provide overlap for data transfers. It is also possible to have independent progress without overlap. An example of this is the implementation of MPI for ASCI Red [6], where the interrupt-driven nature of the network interface insures that progress is made, but the host processor is dedicated to moving data to the network (at least in the default mode of operation where the communication co-processor is not used). As with overlap, quantifying the ability of a network and MPI implementation to provide independent progress is

straightforward, but quantifying the potential of an application to take advantage of independent progress is more complex.

This study is similar to previous work that characterizes the message passing behavior of applications and application benchmarks in an attempt to understand or predict performance. Examples of this type of analysis can be found in [16] and [17]. Rather than just reporting on observed behavior, our work in this paper pinpoints specific characteristics and their impact on performance. Our work is also similar to work such as [9] that compares micro-benchmark and application benchmark performance for various networks. However, we are interested in characterizing an application's ability to leverage network performance rather than simply evaluating and comparing network performance.

3. ANALYSIS METHODOLOGY

The goal of this work is to quantitatively separate, as much as possible, the respective advantages of overlap, independent progress, and offload for a network and for a given MPI library. Doing so requires multiple MPI implementations having different capabilities with comparable micro-benchmark performance on identical platforms. This section first describes the platforms that were used for analysis and then describes the MPI implementations that were evaluated and their contribution to the analysis.

3.1 Platforms

Results were gathered on two platforms: a Linux cluster with a commodity high-performance network and a large-scale massively parallel processing system with a proprietary interconnect. The following describes the hardware and software components of these systems.

The Linux cluster used for our experiments is a 32-node cluster at Los Alamos National Laboratory. Each node in the cluster contains two 1 GHz Intel Itanium-2 processors, 2 GB of main memory, and two Quadrics QsNet (ELAN-3) network interface cards. The nodes were running a patched version of the Linux 2.4.21 kernel. All applications were compiled using Version 7.1 Build 20031106 of the Intel compiler suite.

Compared to a traditional Linux cluster, ASCI Red[14] is a relatively unique system. It is a large-scale supercomputer comprised of more than 4500 dual-processor nodes connected by a high-performance custom network fabric. Each compute node has two 333 MHz Pentium II Xeon processors. Each compute node has a network interface, called a CNIC, that resides on the memory bus and allows for low-latency access to all of physical memory on a node. This interface is effectively a DMA engine that can be instructed to transfer the contents of messages into memory in up to 4 KB chunks. The CNIC interface connects each node to a 3-D mesh network that provides a 400 MB/s uni-directional wormhole-routed connection between the nodes. The CNIC interface is capable of sustaining the 400MB/s node-to-node transfer rate to and from main memory across the entire machine.

The software environment on ASCI Red is also significantly different from the standard commodity model. The compute nodes run Cougar, a variant of the Puma lightweight kernel that was designed and developed by Sandia and the University of New Mexico for maximizing both message passing throughput and application resource availability [13].

Cougar uses a simple network protocol built around the Portals message passing interface [13]. Portals are data structures in an application's address space that determine how the kernel should respond to message passing events. Portals allow the kernel to deliver messages directly from the network to the application's memory.

Cougar is not a traditional symmetric multi-processing operating

Table 1: MPI Implementation Characteristics

	ASCI Red				Quadrics	
	Eager		Rendezvous		Tports	SHMEM
	P0	P1	P0	P1		
Progress	✓	✓			✓	
Overlap		✓		✓	✓	✓
Offload		✓		✓	✓	

system. Instead, it supports four different modes that allow different distributions of application processes on the processors. The following provides an overview of two of these processor modes that are relevant to this paper. More details can be found in [10].

The simplest processor usage mode is to run both the kernel and application process on the system processor. This mode (P0 mode) is commonly referred to as “heater mode” since the second processor is not used and only generates heat. In this mode, the kernel runs only when responding to network events or in response to a system call from the application process.

In the second mode, message co-processor mode (or P1 mode), the kernel runs on the system processor and the application process runs on the user processor. When the processors are configured in this mode, the kernel runs continuously waiting to process events from external devices or service system call requests from the application process. Because the time to transition from user mode to supervisor mode and back can be significant, this mode offers the advantage of reduced network latency and faster system call response time.

3.2 MPI Libraries

Table 1 summarizes the characteristics of the MPI implementations used in our analysis. On the Quadrics cluster, two versions of MPI software were used: the default MPICH variant from Quadrics using the Tports API (version 1.24-27) and a variant of MPICH 1.2.5 built at Sandia [2] using the Cray SHMEM API [7] (using version 1.4.12-1 of the QsNet libraries that contained the Cray SHMEM compatibility library). The MPI version built using the SHMEM API performs quite well as seen in Figure 1; thus the only appreciable difference in the two libraries is their capabilities. Tports provides MPI with the capability to overlap computation and communication, the ability to achieve independent progress, and the ability to offload matching semantics to the network interface. In previous work, it was found that the combination of these factors had a significant impact on the performance of applications [5].

Unfortunately, using only these two implementations on the Quadrics network makes it quite difficult to distinguish the actual source of this performance improvement. ASCI Red offers a platform where finer distinctions can be made. On ASCI Red, the default MPI (based on MPICH 1.0.12) uses interrupt driven Portals processing in the Cougar kernel to provide independent progress. MPI only needs to setup the appropriate Portal for a receive operation and a matching put will be placed directly into user memory without application intervention. This is only possible because the sender uses an eager protocol for all messages. Long messages that arrive without a matching posted receive place MPI envelope information into a buffer and discard the data (to be retrieved later when the matching receive is posted). As part of the interrupt driven processing, the processor is kept busy (in the kernel) for the entire message arrival time because the kernel must continually service the CNIC until the entire message has been received.

Using various processor modes, the default MPI library can pro-

vide a variety of capabilities to the application. In P0 mode, message reception occurs on the application processor. Thus, the MPI library can provide independent progress, but not overlap or offload. In P1 mode, message reception happens on the message co-processor, so the MPI library can provide overlap and offload as well as independent progress.

The primary additional requirement for ASCI Red is a baseline for comparison. An implementation of MPI using a rendezvous protocol for long messages was developed as part of previous work [3] for this purpose. When using a rendezvous protocol in P0 mode, MPI is unable to provide overlap, independent progress, or offload. This library can also be used in P1 mode to test a design point without independent progress, but with overlap and offload capabilities.

4. RESULTS

This section presents the performance of various MPI implementations as a baseline for comparison. It then compares the performance of `MPI_Alltoallv`, which is the core communication routine from the IS NAS parallel benchmarks. Then, the performance of three of the NAS parallel benchmarks when using these MPI implementations is analyzed. By comparing various MPI implementations on the same platform, performance differences are attributable to offload, overlap, or independent progress.

4.1 Microbenchmarks

Figure 1 is included to show the relative performance of the various MPI implementations on standard microbenchmarks. In almost every case, the implementation that does not provide overlap or independent progress appears to be of approximately equal performance to the one that does. The notable exceptions are the latency of the SHMEM interface on ASCI Red and the latency of the rendezvous implementation immediately after it crosses over the “short message” threshold. The erratic behavior of P1 mode on ASCI Red is still being debugged.

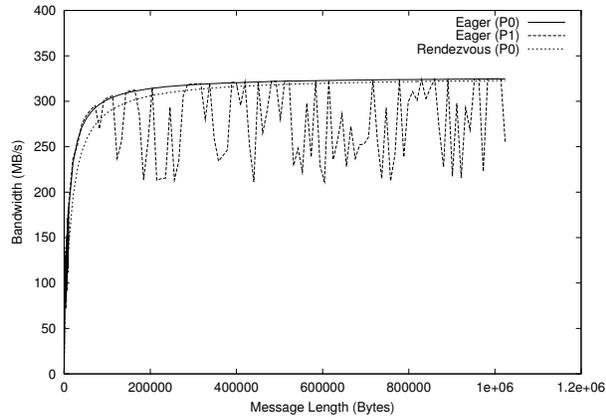
4.2 Collective Benchmarks

The `MPI_Alltoallv` collective is the only significant contributors to communication time for the IS benchmark. As such, the time to perform `MPI_Alltoallv` is graphed in Figure 2 for each of the MPI implementations evaluated. On ASCI Red, there are effectively two classes of performance on this collective: slower performance (longer time) for the default MPI implementation and faster performance (shorter time) for the MPI implementations that use a rendezvous protocol. This is a logical result from previous work that indicated that the naive MPICH implementation of this collective caused a large number of unexpected messages[4] and that the eager MPI implementation that enables independent progress on ASCI Red causes significant performance degradation in the presence of a large number of unexpected messages[3].

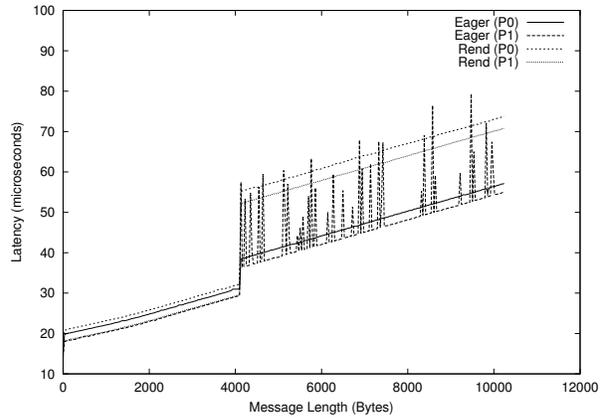
What is more surprising is the data from the Quadrics network. At small message sizes (under 12 KB and not visible on the graph), the SHMEM implementation of `MPI_Alltoallv` holds a slight performance advantage. This is an impact of the slower processor on the Quadrics network interface that must handle the matching semantics[15]. By 12 KB, this advantage has vanished as the overhead of match list traversal is amortized over larger messages. When the message size reaches 1 MB, the Tports implementation is over 30% faster.

4.3 NAS Parallel Benchmarks

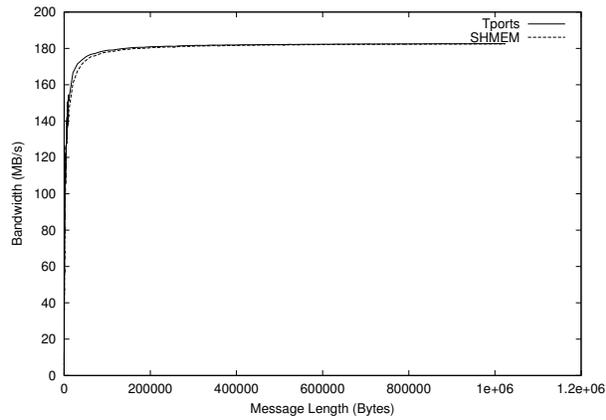
Measurements taken with the NAS Parallel Benchmarks across a variety of MPI implementations on two platforms offer an op-



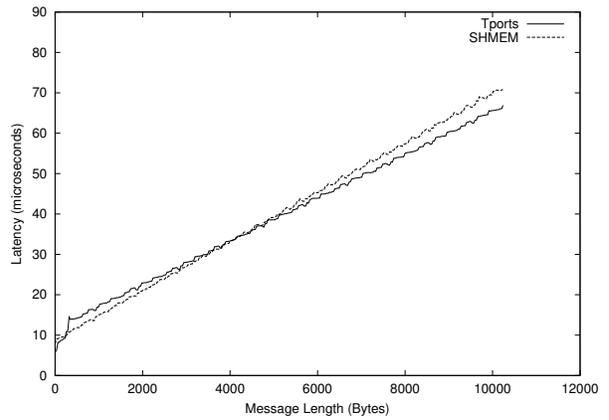
(a)



(b)

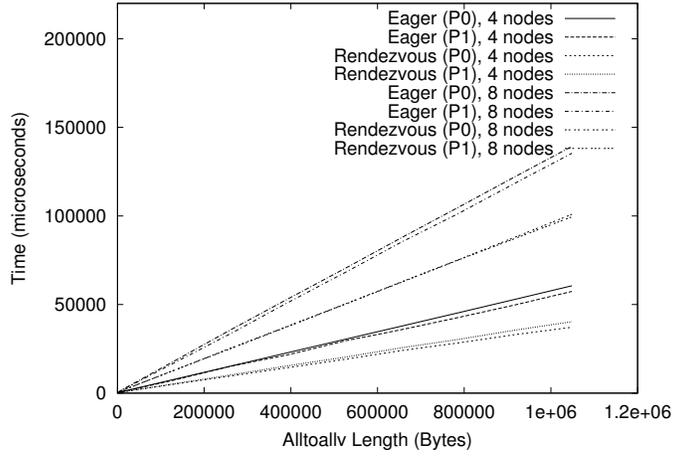


(c)

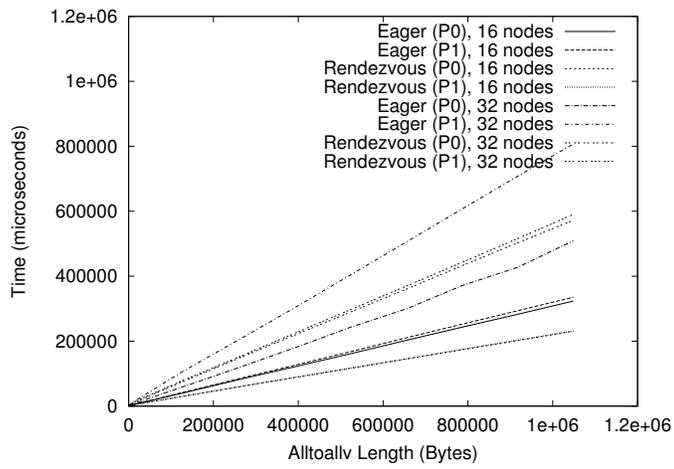


(d)

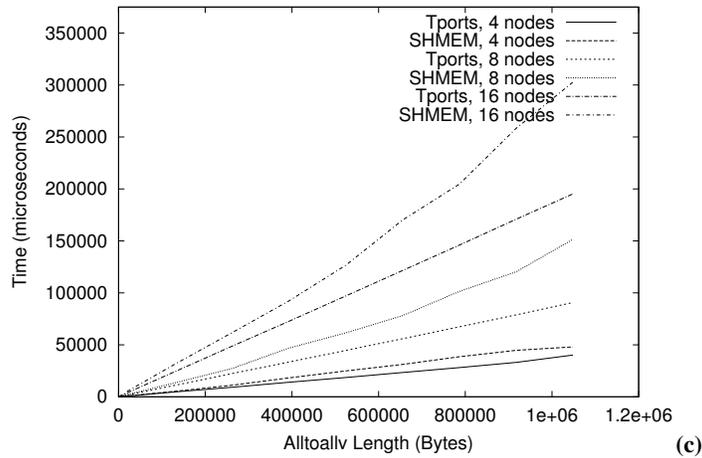
Figure 1: Micro-benchmark comparisons of MPI implementations for ASCI Red ((a) and (b)) and Quadrics Elan3((c) and (d))



(a)



(b)



(c)

Figure 2: MPI Alltoallv performance comparisons of MPI implementations for ASCI Red ((a) and (b)) and Quadrics Elan3 ((c))

portunity to attempt to separate the impacts of overlap, independent progress, and MPI offload. Figure 3 shows data for the IS, SP, and BT class B benchmarks taken from a Quadrics cluster and the ASCI Red platform. These three benchmarks are the only ones that demonstrated a significant performance difference between the SHMEM and Tports implementations of MPI[5]. Each of the measurements presented here is the average of four runs. The measurements from ASCI Red were extremely stable (varying by less than 0.5%); thus, even small differences can be directly attributed to a cause.

Figure 3(a) indicates that Tports outperforms SHMEM by a significant margin on the IS benchmark. This measurement was taken without using the enhanced collectives provided by the default Quadrics MPI (for fairness), but using the enhanced collectives provided virtually no benefit for IS. This is somewhat (but not completely) explained by the significant difference in the `MPI_Alltoallv` performance differences seen in Figure 2. Using Figure 3(b), this difference can be explained by independent progress. Doing so requires a bit of insight. First, the advantages of offload and overlap can be excluded by noticing that P0 performance and P1 performance are virtually identical. Second, contrary its appearance, Figure 3(b) does not contradict the conclusion that the performance advantage Tports over SHMEM is independent progress. This is because the independent progress mechanism on ASCI Red is an eager protocol which experiences a significant reduction in bandwidth in the presence of unexpected messages[3] (as seen in the IS benchmark[4]). This is further reinforced by the data from the rendezvous library run in P1 mode. The addition of overlap and offload to the rendezvous library (without independent progress) adds no performance improvement.

The benefit of independent progress for IS are counterintuitive since the primary communications routine in the IS benchmark is a collective operation; however, the Rogue OS effect[12] is known to cause interference with collective operations. Specifically, in this case, it is key that the independent progress is independent of the application process altogether. Although this is achieved with offload for the Quadrics Tports interface, it can also be achieved with interrupt driven progress, as it is on ASCI Red; thus, it is independent progress, not offload, that affects the performance of the IS benchmark.

The SP and BT benchmarks are more clear cut, especially as the applications scale in the number of nodes. The SP and BT benchmarks using MPI over Tports consistently outperforms MPI over SHMEM on the Quadrics network. The advantage is 3-6% in overall execution time for SP and 2-4% in overall execution time for BT. Using the 64 node runs¹ from ASCI Red, SP sees a fractional gain from adding independent progress (moving from rendezvous to an eager implementation). It sees another 1.5% gain when moving from P0 to P1 mode. This is the combined advantage of adding protocol offload and adding overlap. Notably, when the rendezvous implementation is run in P1 mode, overlap and offload offer less than 1% gain. This is a clear indication that independent progress needs overlap and offload to achieve maximal performance and vice versa. Similarly, BT sees a fractional gain from independent progress but an additional 2.5% gain from the combination of offload and overlap. Again, overlap and offload alone offer only a portion of the advantage.

Moving to 121 nodes changes the picture slightly. At 121 nodes, most of the messages use the short message protocol. Thus, the rendezvous protocol reaps many of the advantages of independent

¹These runs have more comparable execution times to those on the Quadrics cluster and therefore more comparable balances of communication and computation.

progress without truly providing it. Indeed, when adding only independent progress, the BT benchmark actually slows down; however, the combination of independent progress, overlap, and offload still offers a 0.5% advantage over overlap and offload alone.

The most important result here is that overlap must be combined with independent progress to achieve the best results. Independent progress alone suffers from issues such as cache pollution and context switch overheads. Offload and overlap alone suffer because they must wait for the application to return to the MPI library to progress communications. The combination of the three enables a measurable impact on application performance.

5. CONCLUSIONS

This paper has quantitatively explored the advantages of overlap, independent progress, and offload using the NAS parallel benchmarks. Combined, they offer a 2-20% advantage in overall execution time depending on the benchmark. This is particularly significant in the context of \$100 million acquisitions such as those found in the ASCI program. In such cases, a 2% improvement in overall execution time justifies the expenditure of almost \$2 million.

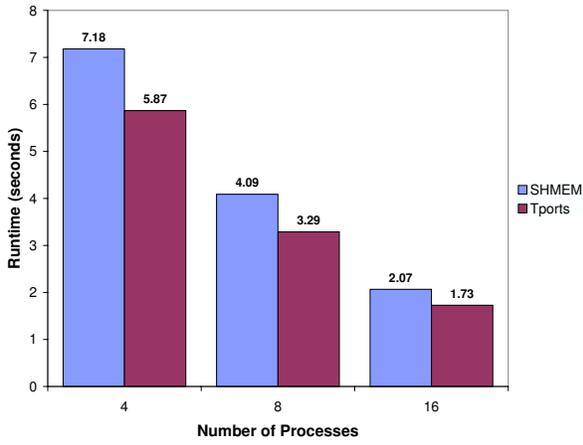
Taken independently, independent progress is a critical contributor to application performance improvements. For both BT and SP, independent progress improves performance as seen on ASCI Red and the Quadrics network. The striking result is that the combined performance improvement from independent progress, overlap, and offload is significantly more than simply the composition of the parts. Another important observation comes from the IS benchmark. The results for IS are a dramatic improvement of 20% on the Quadrics network, but an actual loss of performance on the ASCI Red system. This highlights an important aspect for implementers of independent progress for MPI: the implementation must be careful not to sacrifice too much performance for non-optimal applications (such as those like IS that have a significant number of unexpected messages).

6. FUTURE WORK

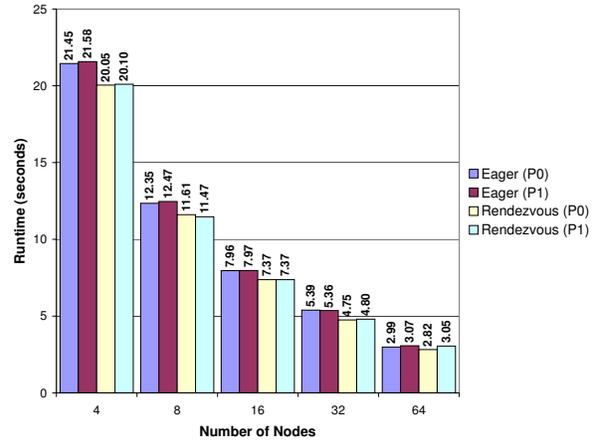
There are three avenues of research that follow from this work. The first is to explore the reasons that overlap, independent progress, and offload offer improved performance. While the mechanism for overlap to offer improved performance is obvious, the reasons for independent progress and offload to offer realizable performance improvement is less clear. The second avenue to follow is to develop measurement techniques that help to further separate the benefits of offload and overlap. The final step in this work is to extend this analysis to applications in the typical workload at Sandia. Such applications are known for taking significant effort to configure, build, and run; hence, their analysis was contingent upon the justification provided by this initial analysis using the NAS parallel benchmarks.

7. REFERENCES

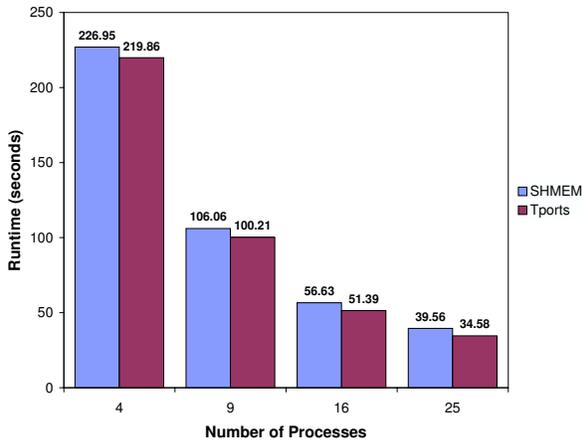
- [1] N. J. Boden, D. Cohen, R. E. F. A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.
- [2] R. Brightwell. A new MPI implementation for Cray SHMEM. Technical report, Sandia National Laboratories. Work in progress.
- [3] R. Brightwell and K. Underwood. Evaluation of an eager protocol optimization for MPI. In *Proceedings of EuroPVM/MPI*, September 2003.
- [4] R. Brightwell and K. D. Underwood. An analysis of NIC resource usage for offloading MPI. In *Proceedings of the*



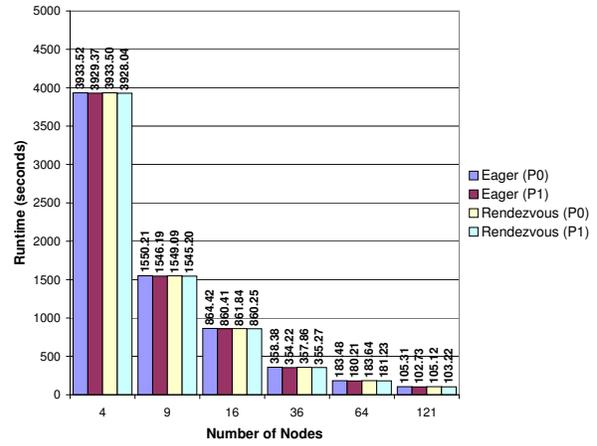
(a)



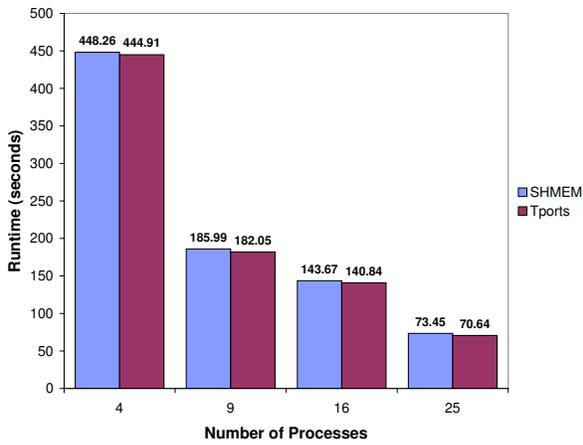
(b)



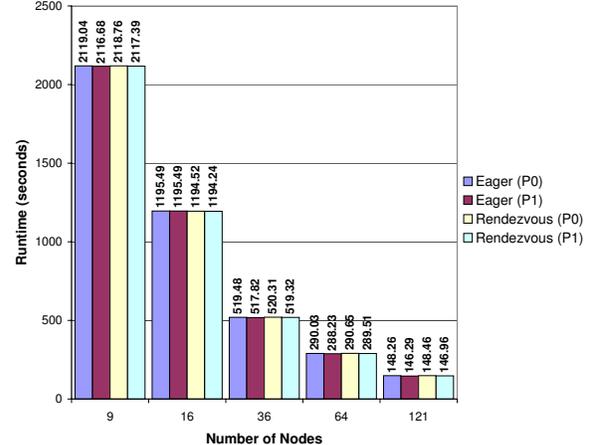
(c)



(d)



(e)



(f)

Figure 3: NAS parallel benchmark comparisons of MPI implementations for Quadrics Elan3: (a) IS, (c) SP, and (e) BT, and ASCII Red: (b) IS, (d) SP, and (f) BT

2002 Workshop on Communication Architecture for Clusters, Santa Fe, NM, April 2004.

- [5] R. Brightwell and K. D. Underwood. An initial analysis of the impact of overlap and independent progress for mpi. In *submitted*, 2004.
- [6] R. B. Brightwell and P. L. Shuler. Design and implementation of MPI on Puma portals. In *Proceedings of the Second MPI Developer's Conference*, pages 18–25, July 1996.
- [7] Cray Research, Inc. *SHMEM Technical Note for C, SG-2516 2.3*, October 1994.
- [8] Infiniband Trade Association. <http://www.infinibandta.org>, 1999.
- [9] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics. In *The International Conference for High Performance Computing and Communications (SC2003)*, November 2003.
- [10] A. B. Maccabe, R. Riesen, and D. W. van Dresser. Dynamic processor modes in Puma. *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)*, 8(2):4–12, 1996.
- [11] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, January/February 2002.
- [12] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Identifying and eliminating the performance variability on the ASCI Q machine. In *Proceedings of the 2003 Conference on High Performance Networking and Computing*, November 2003.
- [13] L. Shuler, C. Jong, R. Riesen, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup. The Puma operating system for massively parallel computers. In *Proceeding of the 1995 Intel Supercomputer User's Group Conference*. Intel Supercomputer User's Group, 1995.
- [14] S. R. W. Timothy G. Mattson, David Scott. A TeraFLOPS Supercomputer in 1996: The ASCI TFLOP System. In *Proceedings of the 1996 International Parallel Processing Symposium*, 1996.
- [15] K. D. Underwood and R. Brightwell. The impact of mpi queue usage on mpi latency. In *submitted*, 2004.
- [16] J. S. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, pages 27–29, April 2002.
- [17] F. Wong, R. Martin, R. Arpaci-Dusseau, and D. E. Culler. Architectural requirements and scalability of the NAS parallel benchmarks. In *Proceedings of the SC99 Conference on High Performance Networking and Computing*, November 1999.