An Application Based MPI Message Throughput Benchmark

Brian W. Barrett and K. Scott Hemmert
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1319
e-mail: bwbarre@sandia.gov, kshemme@sandia.gov

***Abstract*—Recent trends in high performance computing have renewed interest in the ability of platforms to sustain high message throughput rates. The continued growth in platform scale, combined with emerging application areas, are pushing platforms to support increasing message rates. Best-case message throughput has grown in previous hardware generations due to growing clock rates and software optimization techniques. However, previous work has shown that MPI receive queue length and cache hit rates can drastically impact message throughput, leading to a significantly lower worst-case message throughput. This paper introduces the Sandia Message Throughput benchmark which measures message throughput using a communication pattern which is neither best-case nor worst-case, but which mimics communication patterns found in real-world applications. Results on InfiniBand, Myrinet, and Cray XT platforms are presented, and suggest that message rates on some networks are greatly impacted by cache invalidation between communication phases, simultaneously sending and receiving, and by communicating with more than one peer simultaneously.**

I. Introduction

For HPC systems, interconnect performance has long determined application scalability. As machine sizes are growing to tens of thousands of nodes and beyond, the interconnect plays an even more crucial role in how effectively applications can utilize the compute resources of the machine. The large (and growing) node counts of modern machines are making it increasingly important to properly balance the processing capabilities of large machines with their interconnect performance. Proper balance will ensure machines that are both more cost effective and more energy efficient. To enable this capability it is necessary to be able to quantify interconnect performance and correlate it to application scalability. This, unfortunately, is not an easy process.

The collection of meaningful measurements for interconnect performance can be quite challenging. The first thing to determine is which properties of the interconnect are most important. In this regard, the three most commonly measured metrics are bandwidth, latency and message rate. However, other parameters, such as independent progress, host overhead, etc., can also impact application performance. The more difficult question is: how should the data be measured given that interconnect performance can vary dramatically based on the operating conditions of the application using it? Most modern benchmarks do not account for the impact of application behavior on interconnect performance. As an example, the average length of the MPI message queues will impact both the latency and message rate [1] that the network can deliver.

Of the three main measures of interconnect performance, message rate seems to be receiving the most attention lately. Message rate is the measure of how many distinct messages a node (or process) can send and/or receive in a given time period, and is often referred to as message throughput. For small messages, the message rate will determine how much of the available network bandwidth can be effectively utilized; maximum bandwidth able to be utilized is determined by multiplying the message rate and message size. For example, a message rate of 1 million messages per second would only be able to sustain a bandwidth of 8 MB/s for messages of size 8 bytes and a bandwidth of 1 GB/s for messages of 1 KB. Thus, the message rate determines the minimum message size which can saturate the bandwidth of a given network. This effect creates the typical "S-curve" pattern generated by streaming bandwidth benchmarks.

With increasing network bandwidths, message rate is becoming ever more important and interconnect architectures such as InfiniPath [2] are placing increased emphasis on high message rate. Recent advances have pushed raw MPI message rates to close to 10 million messages per second. This means that on a 4 GB/second network, one would expect to be able to saturate the bandwidth with 400 Byte messages. Unfortunately, message rate has traditionally been measured in idealized conditions, not realizable during typical application runs, making the measured message rates over-optimistic. More importantly, both hardware and software vendors have optimized their products to the prevailing microbenchmarks, but it is unclear if these optimizations offer any real advantage to applications.

This paper describes the Sandia Message Throughput Benchmark, which tests message rate in conditions more closely resembling those found during application runs. It does this by mimicking the computation and communication phases generally found in scientific applications. These communication patterns are described in Section III. Section II will describe related work and provide background information. Data from running the benchmark on three platforms, each using a different type of interconnect, will be presented in Section IV, and conclusions are found in Section V.

978-1-4244-5012-1/09/$25.00 ©2009 IEEE

II. Background

Interconnection networks in modern HPC systems have a variety of performance parameters which impact application performance. Many metrics have been developed to try to describe this performance. One attempt to model these parameters and access their impact on applications led to the development of the LogP model [3], [4]. Follow-up work incorporating an analysis of long messages generated the LogGP [5] model. More recent work has developed techniques for measuring the LogGP parameters on modern networks [6]. In addition to these modeling efforts, there is a body of work on benchmarks to measure various aspects of interconnect performance. NetPIPE [7], and Netperf [8], Ohio State's OMB [9], and Intel's MPI Benchmarks [10] can be used to measure network latency and bandwidth.

In addition to these general measures, more specific work has been done to quantify MPI performance. The OSU benchmark suite [9] has microbenchmarks for measuring latency, streaming bandwidth and message rate, among others. Further research has worked to measure additional areas of interconnect performance such as overlap [11]. While others have looked at the impact of queue lengths [1] and overhead and buffer re-use [12].

Unfortunately, many of the network microbenchmarks measure performance in idealized conditions that do not match those present during application execution. Additionally, it is not uncommon for hardware and MPI developers to optimize to the most common microbenchmarks. Many areas where such optimizations improve microbenchmark performance, but have little to no impact on application performance have been identified in [13]. One issue identified with traditional microbenchmarks is that the only operation performed is the sending and receiving of data, which means that the MPI data structures are always in cache. This is not the typical operating environment for real applications, which will intersperse communication with computation.

Another area of specific concern is message coalescing. Both Open MPI [14] and MVAPICH [15] coalesce short messages when running with software flow-control. There are many ways to coalesce messages; the simplest method implemented today works only for zero-byte messages with identical MPI envelope information. However, for coalescing to be generally useful to applications, it must work on messages with data sizes greater than zero and across messages with non-identical tags.

III. Benchmark

The Sandia Message Throughput Benchmark is designed to measure message rate under conditions mimicking those found in typical scientific applications. This results in a benchmark that tests a set of three different communication patterns, all of which are modeled after common application patterns. In particular, pair-based communication, pre-posted receives, and an all-start model are tested. A single-direction pair-based test which is similar to the communication pattern found in most message rate benchmarks is also provided to offer comparison and validation with other existing benchmarks. All tests share a number of features in common, including a computation/communication phase design, variable tags in communication, and the ability to send data (rather than using 0-byte messages) during communication. In addition, the reported message rate for each process includes both the sends and receives accomplished by that process.

Many large-scale applications can be divided into periods of computation and communication, which repeat for the life of the application. During the computation phase, a significant portion of main memory, at least as large as the cache available on modern processors, is modified, resulting in cache misses during communicate phases. Each test includes a cache invalidation step, which simulates an application's working set for each computation phase. Previous work has shown that communication performance, particularly when MPI message matching occurs on the main processor, is sensitive to cache hit rates. An application is likely to invalidate cache lines holding MPI structures during the computation phase, as any real working set is likely at least as large as cache. Unlike previous work, however, our message rate benchmark issues multiple communication requests in a single communication phase, allowing for some cache-use efficiency during communication.

In order to prevent benchmark optimizations, such as aggressive message coalescing, from skewing message rates from those achievable in an application, the benchmark both changes tags for each message and optionally sends a payload from application memory in each message. The size of the messages transferred is a run-time configurable option, and defaults to 8 bytes. The sending of a payload also exposes the impact of transferring to/from host memory during communication, which greatly impacts both latency and message rate for some networks.

A. Single-Direction Communication

The single-direction communication test mimics existing message rate benchmarks. Unlike the other tests presented in this section, we do not believe this test is representative of any real-world application. However, in our own validation during benchmark development, it was useful to verify our benchmark against existing message rate benchmarks.

Processes are paired off, with the lower rank sending messages to the higher rank in a tight loop. The individual pairs synchronize before communication begins to minimize jitter in measurements. Process pairs are chosen to minimize the number of pairs placed on the same node and maximize traffic across the network. Fig. 1 presents psuedo-code for the benchmark kernel. The loop is slightly reordered in the benchmark to reduce the number of conditionals in the timing block, but the communication pattern is preserved.

B. Pair-based Communication

In the pair-based communication pattern, shown in Fig. 2, each process communicates with a small number of remote processes (variable as a configuration parameter) in each communication phase. The communication is paired, so that

```
for (i = 0 ; i < niters ; ++i) {
  nreqs = 0;

  cache_invalidate();
  synchronize();
  start = timer();
  if (rank < size / 2) {
    for (k = 0 ; k < nmsgs ; ++k) {
      MPI_Isend(send_buf + (nbytes * k),
      nbytes, MPI_CHAR, rank + (size / 2), tag,
      comm, &reqs[nreqs++]);
    }
  } else {
    for (k = 0 ; k < nmsgs ; ++k) {
      MPI_Irecv(recv_buf + (nbytes * k),
      nbytes, MPI_CHAR, rank - (size / 2), tag,
      comm, &reqs[nreqs++]);
    }
  }
  MPI_Waitall(nreqs, reqs,
            MPI_STATUSES_IGNORE);
  total += (timer() - start);
}
```

Fig. 1. Message rate benchmark with single-direction communication. Each process either sends or receives messages with exactly one other peer.

```
for (i = 0 ; i < niters ; ++i) {
  cache_invalidate();
  MPI_Barrier(MPI_COMM_WORLD);
  start = timer();
  for (j = 0 ; j < npeers ; ++j) {
    nreqs = 0;
    for (k = 0 ; k < nmsgs ; ++k) {
      offset = nbytes * (k + j * nmsgs);
      MPI_Irecv(recv_buf + offset,
              nbytes, MPI_CHAR,
              recv_peers[j], tag,
              MPI_COMM_WORLD, &reqs[nreqs++]);
    }
    for (k = 0 ; k < nmsgs ; ++k) {
      offset = nbytes * (k + j * nmsgs);
      MPI_Isend(send_buf + offset,
              nbytes, MPI_CHAR,
              send_peers[npeers - j - 1], tag,
              MPI_COMM_WORLD, &reqs[nreqs++]);
    }
    MPI_Waitall(nreqs, reqs,
              MPI_STATUSES_IGNORE);
  }
  total += (timer() - start);
}
```

Fig. 2. Message rate benchmark with pair-based communication. Each process sends and receives messages with a number of remote processes, one at a time.

a given process is both sending and receiving messages with exactly one other process at a time, rotating to a new process when communication is complete. A best effort is made to ensure that the remote processes a given process must communicate with are located on remote nodes, in order to more fully stress the network. This is likely a departure from our application-centric approach, as it is likely that at least one communicating process would be on the same node in a multi-socket/multi-core environment.

Although the test posts non-blocking receives from a given remote process before sending to that process, synchronization can not be guaranteed and the test may result in a number of unexpected messages. It is likely that for a given process, messages will be both expected and unexpected, although the ratio is unknown and likely to vary between tests. Unlike the other application-based tests discussed later, the pair-based communication only posts `nmsgs` receives at a time, meaning the test has the smallest number of outstanding receives at a given time, which may be beneficial on platforms with hardware support for message processing.

C. Pre-posted Communication

In order to increase the probability of expected message reception, a number of applications pre-post receives for the next communication phase before starting the computation phase. The long computation phase essentially guarantees that receive buffers will be available during the communication phase and that all messages are expected by the MPI layer. On networks with hardware matching, this communication paradigm can be especially useful, as the number of memory copies, and therefore wasted memory bandwidth, is reduced to a minimum.

The pre-posted communication test simulates such a model by posting `nmsgs` receives from all communicating processes, invalidating the cache to simulate the computation working set and synchronizing with a barrier, followed by starting `nmsgs` sends to all communicating processes. Although the test guarantees that all receives are posted, it also tends to push the receive queue out of cache for early receives due to the cache invalidation between posting the receives and starting the sends. The number of receives posted at any time is $nprocesses * nmsgs$, which is considerably higher than the pair-based communication test.

D. All-Start Communication

The all-start communication test possesses many of the same properties as the pre-posted communication test, but does not guarantee that all receives are pre-posted and invalidates the cache to simulate the computation working set before any communication calls in a given iteration. The test simulates an application which finishes a computation phase, then issues all communication calls at once with a single `MPI_WAITALL` call to complete all communication.

Like the pre-posted communication test, the MPI is forced to deal with a large number of outstanding receives. The test will also likely cause the MPI to have to search a large portion of the expected queue for any incoming message, as the queue is ordered by remote process. MPI implementations which optimize queue searching by maintaining per-process receive queues in addition to a global queue for handling `MPI_ANY_SOURCE` may be able to avoid the deep queue search. Like the per-pair communication test, the MPI is likely

```
start = timer();
for (j = 0 ; j < npeers ; ++j) {
  for (k = 0 ; k < nmsgs ; ++k) {
    offset = nbytes * (k + j * nmsgs);
    MPI_Irecv(recv_buf + offset
            nbytes, MPI_CHAR,
            recv_peers[j], tag,
            MPI_COMM_WORLD, &reqs[nreqs++]);
  }
}
total += (timer() - start);

for (i = 0 ; i < niters - 1 ; ++i) {
  cache_invalidate();
  MPI_Barrier(MPI_COMM_WORLD);

  start = timer();
  for (j = 0 ; j < npeers ; ++j) {
    for (k = 0 ; k < nmsgs ; ++k) {
      offset = nbytes * (k + j * nmsgs);
      MPI_Isend(send_buf + offset
              nbytes, MPI_CHAR,
              send_peers[npeers - j - 1], tag,
              MPI_COMM_WORLD, &reqs[nreqs++]);
    }
  }
  MPI_Waitall(nreqs, reqs,
            MPI_STATUSES_IGNORE);
  nreqs = 0;
  for (j = 0 ; j < npeers ; ++j) {
    for (k = 0 ; k < nmsgs ; ++k) {
      offset = nbytes * (k + j * nmsgs);
      MPI_Irecv(recv_buf + offset
              nbytes, MPI_CHAR,
              recv_peers[j], tag,
              MPI_COMM_WORLD, &reqs[nreqs++]);
    }
  }
  total += (timer() - start);
}

start = timer();
for (j = 0 ; j < npeers ; ++j) {
  for (k = 0 ; k < nmsgs ; ++k) {
    offset = nbytes * (k + j * nmsgs);
    MPI_Isend(send_buf + offset,
            nbytes, MPI_CHAR,
            send_peers[npeers - j - 1], tag,
            MPI_COMM_WORLD, &reqs[nreqs++]);
  }
}
MPI_Waitall(nreqs, reqs, MPI_STATUSES_IGNORE);
total += (timer() - start);
```

Fig. 3. Message rate benchmark with pre-posted receives. Each process sends and receives messages with a number of remote processes, posting all receives, then all sends.

to see a mix of expected and unexpected messages, depending on the timing of a particular run of the test.

```
for (i = 0 ; i < niters ; ++i) {
  cache_invalidate();
  MPI_Barrier(MPI_COMM_WORLD);

  start = timer();
  nreqs = 0;
  for (j = 0 ; j < npeers ; ++j) {
    for (k = 0 ; k < nmsgs ; ++k) {
      offset = nbytes * (k + j * nmsgs);
      MPI_Irecv(recv_buf + offset,
              nbytes, MPI_CHAR,
              recv_peers[j], tag,
              MPI_COMM_WORLD, &reqs[nreqs++]);
    }
    for (k = 0 ; k < nmsgs ; ++k) {
      offset = nbytes * (k + j * nmsgs);
      MPI_Isend(send_buf + offset,
              nbytes, MPI_CHAR,
              send_peers[npeers - j - 1], tag,
              MPI_COMM_WORLD, &reqs[nreqs++]);
    }
  }
  MPI_Waitall(nreqs, reqs, MPI_STATUSES_IGNORE);
  total += (timer() - start);
}
```

Fig. 4. Message rate benchmark with all sends and receives started before waiting for completion and no pre-posted guarantees.

IV. Results

A. Experimental Setup

Three platforms with different network designs are used to compare benchmark results: a Cray XT-based architecture, an InfiniBand-based cluster, and a Myrinet-based cluster. While all three platforms utilize the Opteron processor, node design, processor revision, and operating system vary greatly between platforms.

The Red Storm platform installed at Sandia National Laboratories was the predecessor to the Cray XT line of platforms. The test environment used for this paper consisted of 24 nodes with a single 2.4 GHz dual-core Opteron processor and 48 nodes with a single 2.2 GHz quad-core Opteron processor (all benchmarks were run exclusively on the quad-core nodes). Each node utilizes a single SeaStar NIC capable of 4 GB/s bidirectional bandwidth, configured in a mesh topology. The system was configured with the Unicos/lc operating system based on the Catamount lightweight kernel, version 2.0.62. Cray's MPI, derived from MPICH2, was used for testing.

The InfiniBand-based cluster is 272 nodes, each with four quad-core 2.2 GHz Opteron processors. Each node contains a single Mellanox ConnectX HCA connected to a Voltaire DDR switch. The system runs a variant of Red Hat Enterprise Linux, Open Fabrics Enterprise Edition 1.2, and Open MPI 1.2.7.

The Myrinet-based cluster has 128 nodes, each with two single-core 2.2 GHz Opteron processors. Each node contains a single Myrinet 10G NIC (10G-PCIE-8A-C) connected to a single Myrinet 10G switch. The system runs Red Hat Enterprise Linux 4, Update 7, Myrinet MX 1.2.7, and Open

MPI 1.2.8. Open MPI was configured to use the native MX matching, rather than the default of MPI matching inside the Open MPI library.

Unlike Red Storm and the InfiniBand hardware used for this experiment, the Myrinet hardware used is capable of NIC-based adaptive routing. [16] The number of messages "in flight" at any given time is constant and well under theoretical network limits for any experiment, therefore we believe the trends presented in this section are related to endpoint and software design more than the ability to adaptively route messages.

All tests were executed on 32 nodes, with the number of processes in the job varied based on the desired number of processes per node. The number of iterations was set to 4096, the number of messages to each peer in an iteration at 128, and the message size at 8 bytes. The number of communicating peers, the number of processes per node, and the working set size simulated were all varied during testing.

B. Working Set Analysis

Figs. 5, 6, and 7 examine the achievable message rate when larger blocks of cache are invalidated by reading and writing to application memory regions. As the cache invalidation simulates an application's computation phase, its impact is important to understanding the message rates an application is likely to experience. One process is placed on each node, and the message rate is the average of all 32 processes. We use the average rather than the maximum as we are more interested in overall application impact, rather than nearest neighbor artifacts. For the pair-based, pre-posted, and all-start communication patterns, every process communicates with six other peers. In the single direction test, each process communicates with exactly one other process.

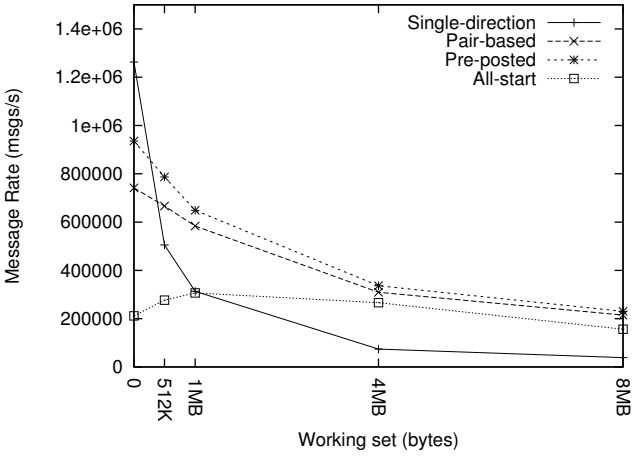

Fig. 5. Impact of working set size on message rate on a InfiniBand interconnect.

InfiniBand provides extremely high message rates when MPI structures are not pushed out of cache by application data (Fig. 5). However, as a greater amount of MPI structure is pushed out of cache into main memory, the InfiniBand message rates generally suffer. The cases where processes both send and receive result in a lower message rate than when a process is only sending or receiving (the single-direction case), although they also appear to be less severely impacted by cache misses. This is likely because there is more work between cache invalidation, due to both a greater number of peers and double the operations per peer (both sending and receiving), amortizing the cost of loading MPI structures back into cache. The InfiniBand result suggests that benchmarks which test the far left side of the graph for the single-direction communication pattern produce results which do not represent achievable rates in a real application.

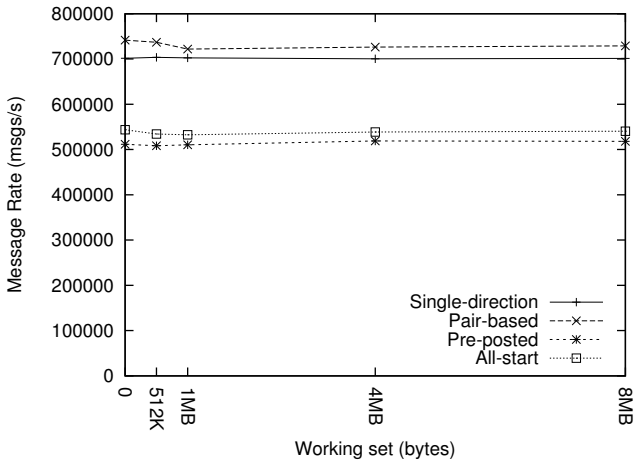

Fig. 6. Impact of working set size on message rate on a Myrinet 10G network.

Myrinet 10G is unable to match InfiniBand for peak message rate. However, Myrinet's message rate is not significantly impacted by working size set, which leads to message rates which are higher than those provided by InfiniBand for working sets over 1 MB in size. For real applications, 1 MB would be an extremely small data set, suggesting that Myrinet may provide better performance for message rate bound applications.

The message rate suggests that Myrinet is able to sustain a higher message rate when communication occurs with only one peer at a time, as the single direction and pair-based communication tests offer a higher message rate than the pre-post and all-start messaging tests. They also suggest that Myrinet is not as sensitive to unexpected messages as to number of peers actively communicating. Discussions with the author of the Myrinet software interface, MyriExpress [17] suggests a significant reason for the ability to sustain performance in the face of working set size is a careful combination of cache-aware data structure and interrupts to wake a processing thread when the host is not actively processing network traffic. The cache-aware data structures ensure that numerous receive queue entries are brought into cache on the first load, mitigating the cost of the first cache miss.

Unlike InfiniBand and Myrinet, the Cray platform and its SeaStar network perform matching in kernel space. Posting a

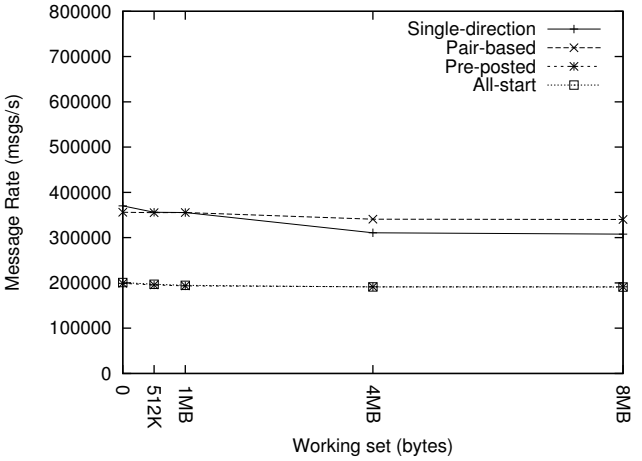

Fig. 7. Impact of working set size on message rate on a Cray XT SeaStar network.

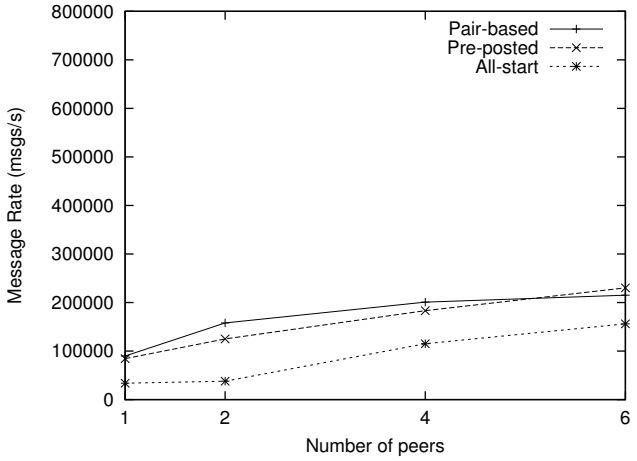

Fig. 8. Impact of peer count on message rate on an InfiniBand network.

send or receive involves a trap to the kernel, and incoming messages result in an interrupt on the host processor. The result is the lowest message rate of the three networks (Fig. 7), although it is relatively insensitive to working set size. It is believed that handling incoming messages with an interrupt handled in kernel space is likely the cause of the low working set influence.

Similar to MX, although the message rate is uniformally low, it is still higher than InfiniBand's rates for large working set sizes. Unlike MX, SeaStar isn't able to match InfiniBand's message rates until the working set size reaches 8 MB, which is the size of cache on most processors, but still not an unreasonable working set for real applications.

C. Peer Count Analysis

The results for pair-based, pre-posted, and all-start communication patterns presented in Section IV-B assume a process is communicating with six other peers. In this section, the impact of varying the number of peers utilized in communication is examined. The working set is held fixed at 8 MB and as with previous tests, one process is executed on each of 32 nodes.

Fig. 8 suggests that InfiniBand is capable of higher message rates as the number of communicating peers increases. The improved performance is likely due to the credit-based flow control utilized within the MPI library. InfiniBand's poor network level flow control behavior requires MPI-level flow control, and the number of credits to a given host is relatively low. With the increased number of peers, the number of available send credits increases.

As Figs. 9 and 10 demonstrate, Myrinet and Cray's SeaStar do not have the same scalability as InfiniBand in message rate as the number of communication peers is increased. Two explanations for the performance behavior are the longer message queues of the pre-posted and all-start tests and a limitation in the network stack when receives are posted for messages from multiple peers. The performance of the pair-based test does not eliminate either hypothesis, as the process only posts receives from a single peer at a time, which also results in a significantly shorter receive queue.

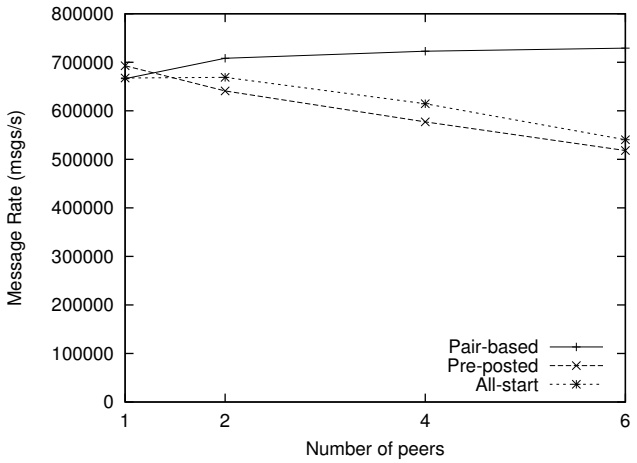

Fig. 9. Impact of peer count on message rate on a Myrinet 10G network.

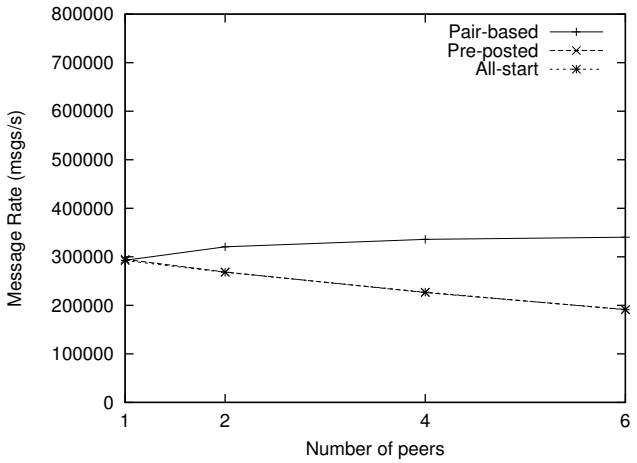

Fig. 10. Impact of peer count on message rate on a Cray XT SeaStar network.

D. Process per Node Scaling

The final results present the impact of running multiple processes on a node, holding the working set size at 8 MB. Rather than raw message rates, the results are presented as scalability based on total node message rate divided by one process per node message rate. The total number of processes in the experiment varies based on the number of process per node, where the node count is held constant at 32. The communication pattern between nodes is setup such that two processes on the same node will not communicate as part of the benchmark measurement.

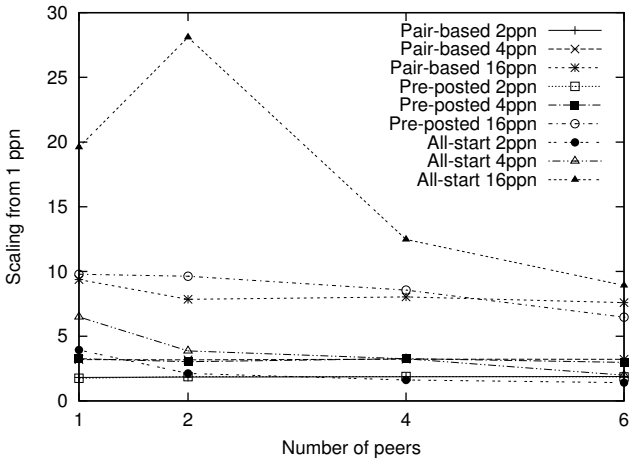

Fig. 11. Process-per-node scalability comparison for InfiniBand networks

Overall, InfiniBand (Fig. 11) shows the best scalability of all three networks. The all-start case is troubling, as it suggests scaling for the 16 processor case greater than 16. The performance for the all-start case on the tested InfiniBand platform is rather noisy and erratic, suggesting an artifact of testing not understood by the authors is resulting in the performance anomaly.

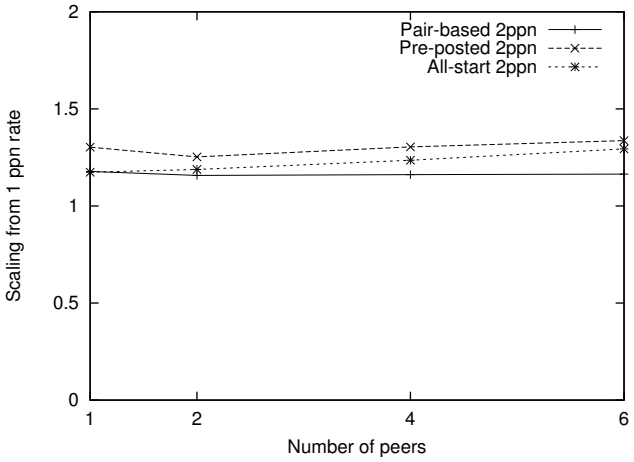

Fig. 12. Process-per-node scalability comparison for Myrinet 10G networks

The test platform for Myrinet (Fig. 12) only provides two single core processors, limiting the scalability test to the two process per node case. The overall scalability of the network is poor, with aggregate message rates only increasing 30% when the number of processes on a node is doubled. Such a result is somewhat surprising, as matching occurs on the host processor with Myrinet, and message processing on the receive side is generally the limiting factor. However, the older communication architecture of the platform rather than Myrinet may be to blame, limiting the throughput from both processors to the NIC.

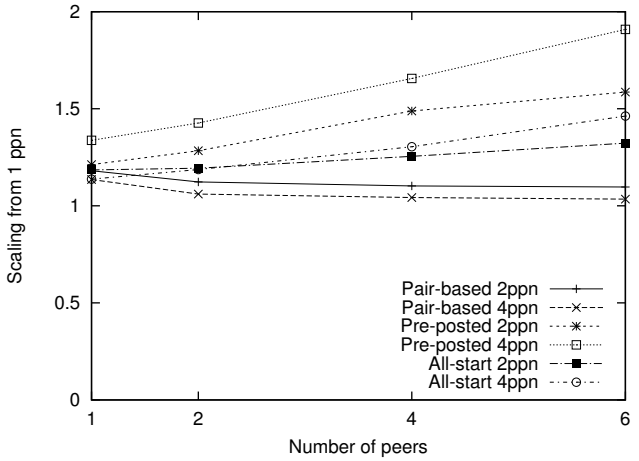

Fig. 13. Process-per-node scalability comparison for Cray XT SeaStar networks

The Cray SeaStar results show poor scalability, with little gain in message rate between two and four processes per node. Such a result is expected, as all incoming message processing on the Catamount-based XT platforms is handled by core 0 of the processor, an artifact likely to be removed in future generations of the XT line. The little scalability gain seen for the pre-posted and all-start cases as the number of communicating peers increases suggests the declining scalability seen in Fig. 10 is not as severe as the number of processes on a node is increased.

E. Comparison with Ohio State Benchmarks

Fig. 14 compares the message rate results from the Ohio State MPI Benchmarks Multiple Bandwidth / Message Rate test with the Sandia Message Throughput Benchmark. The Sandia benchmark was configured such that each process communicated with exactly one other process, there was no cache invalidation, and a single-direction communication pattern was used. This somewhat closely approximates the communication pattern for the Ohio State benchmark and offers a valid comparison point, although it is the least interesting data point for the Sandia benchmark.

The results show a small difference in results on the Cray XT and Myrinet platforms, but a large (30%) performance difference on the InfiniBand platform. The results suggest that InfiniBand is sensitive to factors other than those examined in this paper (such as outstanding message counts), and

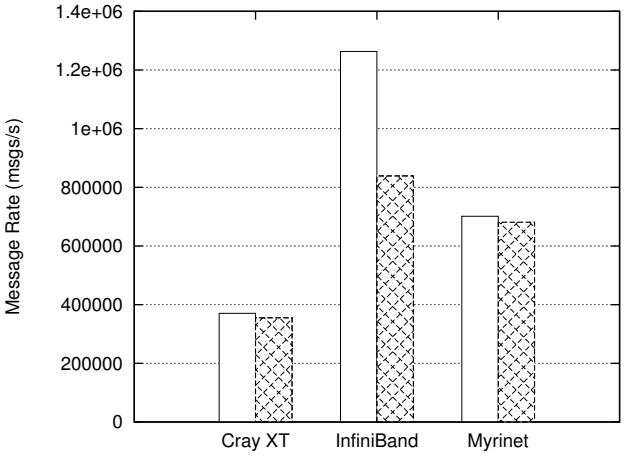

Fig. 14. Comparison between Sandia Message Throughput Benchmark and Ohio State MPI Benchmarks. Sandia results are with one peer, no cache invalidate, and single-direction communication pattern.

the Sandia benchmark is more favorable to InfiniBand in those parameters and suggest further experimentation with the parameters available in the Sandia benchmark would be necessary to more closely approximate the Ohio State Benchmarks, although such an approximation was not a design goal of the benchmark suite.

V. Conclusions

Network design and platform procurment is, in part, driven by performance on micro-benchmarks. Therefore, it is critical that benchmarks accurately reflect the performance characteristics of real applications. This paper presents a new benchmark, the Sandia Message Throughput Benchmark, which tests message rates under scenarios likely to be encountered by real applications by similating an application working set and both sending and receiving data with multiple remote processes. The benchmark is highly parameterizable, allowing in-depth study of conditons which impact acheivable message rates on current networking hardware.

The impact of working set size, number of remote processes involved in communication, and number of processes per node are all examined for InfiniBand, Myrinet, and Cray XT networks. The results suggest that existing message rate benchmarks, which have no application working set simulation, greatly inflate message rates for the InfiniBand network, when compared with achievable rates for most applications. The results also suggest that while both Myrinet and Cray XT hardware have lower message rates in ideal situations, they are more consistent across tests.

The Sandia Message Throughput Benchmark will be made available to the community under the GNU Lesser General Public License (LGPL) as part of the Sandia MPI Micro-Benchmark Suite (SMB) package.

Acknowledgment

The authors would like to thank the reviewers for their advice, as well as Keith Underwood, Kevin Pedretti, and Patrick Geoffray for answering numerous questions. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] K. D. Underwood and R. Brightwell, "The impact of MPI queue usage on message latency," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, Montreal, Canada, August 2004.

[2] L. Dickman, G. Lindahl, D. Olson, J. Rubin, and J. Broughton, "PathScale InfiniPath: A first look," in *Proceedings of the 13th Symposium on High Performance Interconnects (HOTI'05)*, August 2005.

[3] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proceedings 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1993, pp. 1–12.

[4] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, "Effects of communication latency, overhead, and bandwidth in a cluster architecture," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.

[5] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Sheiman, "LogGP: Incorporating long messages into the LogP model," *Journal of Parallel and Distributed Computing*, vol. 44, no. 1, pp. 71–79, 1997.

[6] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, and K. Yelick, "An evaluation of current high-performance networks," in *17th International Parallel and Distributed Processing Symposium (IPDPS'03)*, Apr. 2003.

[7] Q. O. Snell, A. Mikler, and J. L. Gustafson, "NetPIPE: A network protocol independent performance evaluator," in *Proceedings of the IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[8] *Netperf*, http://www.netperf.org.

[9] *Network-Based Computing Laboratory Benchmarks*, http://mvapich.cse.ohio-state.edu/benchmarks/.

[10] *Intel MPI Benchmarks*, http://software.intel.com/en-us/articles/intel-mpi-benchmarks/.

[11] W. Lawry, C. Wilson, A. B. Maccabe, and R. Brightwell, "COMB: A portable benchmark suite for assessing MPI overlap," in *IEEE International Conference on Cluster Computing*, September 2002, poster paper.

[12] 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.

[13] K. Underwood, "Challenges and issues in benchmarking MPI," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 13th European PVM/MPI Users' Group Meeting, Bonn, Germany, September 2006 Proceedings*, ser. Lecture Notes in Computer Science, B. Mohr, J. L. Träff, J. Worringen, and J. Dongarra, Eds., vol. 4192. Springer-Verlag, 2006, pp. 339–346.

[14] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[15] W. Huang, G. Santhanaraman, H.-W. Jin, Q. Gao, and D. K. Panda, "Design and Implementation of High Performance MVAPICH2: MPI2 over InfiniBand," in *International Sympsoium on Cluster Computing and the Grid (CCGrid)*, Singapore, May 2006.

[16] P. Geoffray and T. Hoefler, "Adaptive routing strategies for modern high performance networks," in *16th IEEE Symposium on High Performance Interconnects*, Stan, August 2008, pp. 165 – 172.

[17] Myricom, Inc., "Myrinet Express (MX): A high performance, low-level, message-passing interface for Myrinet," July 2003. [Online]. Available: http://www.myri.com/scs/MX/doc/mx.pdf