What if MPI Collective Operations Were Instantaneous?

Rolf Riesen[1], Courtenay Vaughan[1], and Torsten Hoefler[2]

[1] Sandia National Laboratories*
Albuquerque, NM 87185-1110
rolf@sandia.gov
[2] Technical University Chemnitz
htor@tu-chemnitz.edu

Abstract Collective MPI operations are an interesting research topic, since their implementation is complex and there are many different possible ways to improve their performance. It is not always obvious how much a given application will gain from such improvements and some of the methods to improve collective performance are costly and time-consuming to implement.
In this paper we use a hybrid MPI simulator to run benchmarks and applications with the cost of collective operations set to zero. This allows us to gather performance data that shows how much at most an application could benefit from better collective operations.

Keywords: MPI, collectives, simulation, performance

1 Introduction

Collective operations in MPI provide complex functionality to applications that lets them exchange, gather, and distribute data among the nodes the application is running on. Collective operations are of great interest to system designers because they can be optimized for a given protocol stack, network interface, and network topology without changing the application. Examples of such optimizations include using algorithms that are designed for a specific network topology and off-loading the collective operation into the network interface. Most optimizations for collective operations are complex, difficult to implement, and require time to mature to production quality. Sometimes only a subset of the MPI collective operations are optimized on a given system; e.g. only **MPI_Bcast()**.

It is therefore interesting to know how much application performance could be improved through a given optimization before the time and effort is spent implementing it. In particular, the Cray XT-3 system has a network interface on each node called Seastar. This interface contains DMA logic, a router, memory, and a CPU. Currently the CPU inside the Seastar only handles point-to-point messages. The MPI collective operations are implemented in the host using point-to-point messages.

* 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 Seastar interface is difficult to program and implementing collective operations inside it would require some effort. It is therefore interesting to find out beforehand which applications would benefit from such an effort and by how much. We use a hybrid simulation technique briefly described in the next section to run applications as if collective operations incurred zero cost. Obviously, this is unrealistic but will give us a bound on how much a given application's performance might increase. We want to perform these experiments on several applications and therefore need a measurement technique that requires no changes to the application source code.

2 Hybrid Simulation

We use a hybrid simulator developed at Sandia National Laboratories to conduct our experiments. Under this simulator applications run natively as they normally would, but for each MPI send, receive, and collective operation, an event is sent to a network simulator that runs on one additional node. This is accomplished by linking the application with a wrapper library that makes use of the MPI profiling interface. For each MPI call the library provides a wrapper that manages events and virtual time on each node, and then calls the appropriate PMPI function in the MPI library.

For send and collective operations the wrapper library collects the virtual time of the operation and sends an event to the network simulator. For each receive and collective operation the wrapper library receives information from the network simulator on how long in virtual time that operation took to complete. The library then updates the local virtual time and returns to the application program.

The network simulator can assign arbitrary delay values to each MPI operation. Since the events contain envelope information such as source, destination, tag, and whether it is a point-to-point or a specific collective operation, the simulator can assign different network delays to different types of operations. The application runs in virtual time and does not observe the overhead introduced by the simulator. The simulator is described in more detail in [7].

The current model for simulating collective operations is somewhat simplistic. It does not take into consideration possible congestion in the network and assumes that a logarithmic fan-out and fan-in is used by the MPI library. We will later see that this is basically correct, although actual performance for smaller messages can be quite a bit slower than the model predicts.

The optimistic results reported by the simulator for collective operations limit its usability until we have developed a better model matching the performance profile of the target system. However, since we are only interested in how much applications have to gain if collective operations were completely free of cost, the current model is fine for the experiments described here.

3 Experiments

For our experiments we chose a mixture of benchmarks and applications that use MPI collective routines to perform their work. The more an application depends on collective

operations to distribute and collect data, the more performance increase it should see when we set the cost to do collective operations to zero.

3.1 NAS Parallel Benchmarks

The NAS parallel benchmarks [1] are well understood and offer a simple and quick confirmation that our approach works and produces valid results. Although the NAS parallel benchmarks make use of collective operations, it is mostly to distribute data at the beginning and collect results at the end. For this reason we do not expect a significant performance gain for these benchmarks by reducing the cost of collective operations.

Table 1. Number of collectives used by NAS FT

	Class A			Class B			Class C	
nodes	4	16	64	4	16	64	16	64
Reduce	6	6	6	20	20	20	20	20
Allreduce	2	2	2	2	2	2	2	2
Alltoall	8	8	8	22	22	22	22	22
Alltoallv	0	0	0	0	0	0	0	0
Barrier	1	1	1	1	1	1	1	1
Bcast	6	30	126	6	30	126	30	126

Table 2. Number of collectives used by NAS MG

	Class A			Class B			Class C		
nodes	4	16	64	4	16	64	4	16	64
Reduce	1	1	1	1	1	1	1	1	1
Allreduce	88	88	88	88	88	88	88	88	88
Alltoall	0	0	0	0	0	0	0	0	0
Alltoallv	0	0	0	0	0	0	0	0	0
Barrier	6	6	6	6	6	6	6	6	6
Bcast	18	90	378	18	90	378	18	90	378

Tables 1 and 2 show the number of collective operations used by the FT and MG NAS parallel benchmarks for class A, B, and C runs on 4, 16, and 64 nodes. Both of them use more broadcast operations as the number of nodes being used increases. However, the increase is not significant and the total number of operations remains small.

3.2 All-to-All Benchmark

We wrote a simple benchmark to explore the other end of the extreme. In a loop, the benchmark calls **MPI_Alltoall()** 100 times for messages starting at a length of 4 bytes (a single integer) to 4 MB (one million integers).

Since this benchmark does nothing but collective operations, we expect it to perform significantly better when we set the cost of collective operations to zero.

3.3 Abinit

Abinit [2,3] is a program package which calculates the total energy, charge density and electronic structure of systems composed of electrons and nuclei. It uses a simplification of the Density Functional Theory (DFT) and the solution scheme described in [6] to solve the time independent Schroedinger equation which is in its own an eigenproblem. The conjugate gradient based method proposed by Teter et. al. [8] is used to solve the eigenproblem for this specific scenario. An efficient parallelization of this method is described in [4] and analyzed for its overhead in [5].

The kernel consumes about 98% of the running time and uses MPI-collective operations (**MPI_Alltoall()** and **MPI_Allreduce()**) exclusively to communicate. Work in [5] showed that the scaling of the application is mainly limited by the **MPI_Alltoall()** collective operation. Therefore we expect that Abinit will benefit greatly from improved collective performance.

3.4 Methodology

For each application in the above set we perform three experiments. First we run the application as is to gain a baseline performance number. For the NAS parallel benchmarks we did this on 4, 16, and 64 nodes for the class C benchmarks. The all-to-all benchmark ran on 4, 16, and 64 nodes as well. We ran Abinit on varying numbers of nodes up to 64 to evaluate its scaling.

For the second stage in our experiments we ran each application again, but this time linked together with our simulator. The goal of this stage is to show that the simulator accurately runs the application and that it reports the same (virtual) runtime as in stage one.

Due to the simplistic approach to simulate the collective operations we expect that the runtime of those applications that rely heavily on collective operations will report a better virtual runtime than the actual time reported in stage one.

This limitation means we cannot yet very accurately simulate collectives that have a given performance profile other than zero. However, if we simulate the cost of collectives to be zero, we will get an absolute lower bound of what a given application could achieve in the best, impossible to achieve, case.

In the final stage of our experiments we run the applications under our simulator again and set the cost of collective operations to zero.

These experiments were performed on a Cray XT-3 system named Red Storm at Sandia National Laboratories.This system with nearly 10,000 AMD Opteron nodes was not available to us as a dedicated system. In order to keep our experiments as valid

as possible under these circumstances, we used the batch scheduler to allocate nodes for us and then ran all experiments for a given application on the same set of nodes. This guarantees that the simulated runs used the same nodes and node allocation as the baseline runs.

4 Results

4.1 NAS Parallel Benchmarks

We expected little gain in performance for the NAS parallel benchmarks because these benchmarks use collective operations mainly for synchronization and gathering of results at the end of each run. Tables 3, 4, and 5 show the results of running the class C benchmarks FT, IS, and MG respectively.

We ran each of these three benchmarks on 4, 16, and 64 nodes. On the same set of nodes we ran the benchmarks 7 times as usual (normal), 7 times with the simulator (sim), and 7 times with the network simulator set to incur zero cost for the collectives. The results of the minimum, median, and maximum runtime clearly show that the NAS parallel benchmarks are not at all sensitive to the performance of collective operations.

Table 3. Runtime for NAS FT

	4 nodes			16 nodes			64 nodes		
	min	median	max	min	median	max	min	median	max
normal	218.70	218.96	229.30	59.54	59.74	59.83	15.36	15.42	15.62
sim	218.67	218.92	229.18	59.53	59.69	59.82	15.37	15.43	15.70
zero	218.65	218.89	229.18	59.54	59.63	59.78	15.35	15.42	15.74

Table 4. Runtime for NAS IS

	4 nodes			16 nodes			64 nodes		
	min	median	max	min	median	max	min	median	max
normal	10.99	11.02	15.06	2.73	2.81	4.29	1.45	1.46	2.14
sim	11.00	11.05	15.15	2.77	2.81	4.44	1.45	1.46	2.18
zero	10.98	11.07	15.00	2.77	2.81	4.44	1.41	1.46	2.15

4.2 All-to-All Benchmark

Figure 1 shows the performance of our all-to-all benchmark on 16 nodes, and Figure 2 shows the same benchmark on 64 nodes. The top line shows the irregular performance

Table 5. Runtime for NAS MG

	4 nodes			16 nodes			64 nodes		
	min	median	max	min	median	max	min	median	max
normal	57.59	57.80	57.81	14.53	14.69	14.70	3.40	3.56	3.56
sim	57.59	57.80	57.81	14.54	14.68	14.70	3.40	3.55	3.56
zero	57.59	57.80	57.81	14.53	14.68	14.71	3.40	3.55	3.57

of the benchmark when run in stand-alone mode. The model built into the simulator for collective operations is somewhat optimistical and does not follow exactly the characteristics of the real machine. That behavior is shown in the middle curve.

When we set the cost for collectives to zero, the curve becomes impossible to see, because it is almost level with the x axis.

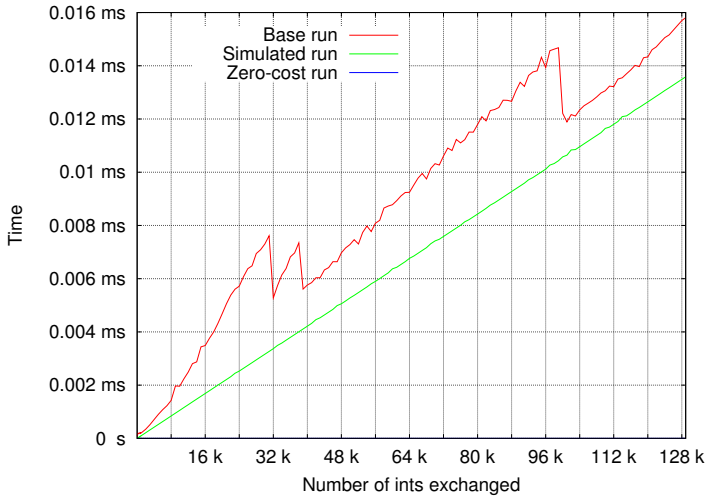

Figure 1. All-to-all on 16 nodes

4.3 Abinit

We show strong scaling results for a small SiO_2 system with 43 atoms, 126 bands, 48728 plane waves and a 61x61x256 FFT grid. (We are currently conducting experiments on a larger SiO_2 system with 86 atoms, 251 bands, 97624 and a 81x81x256 FFT grid, which we will have in time for the camera ready submission.)

Figure 3 shows the results for that simulation. We use a log scale on the y axis to emphasize the result. Again, the top line shows the performance of Abinit when run in

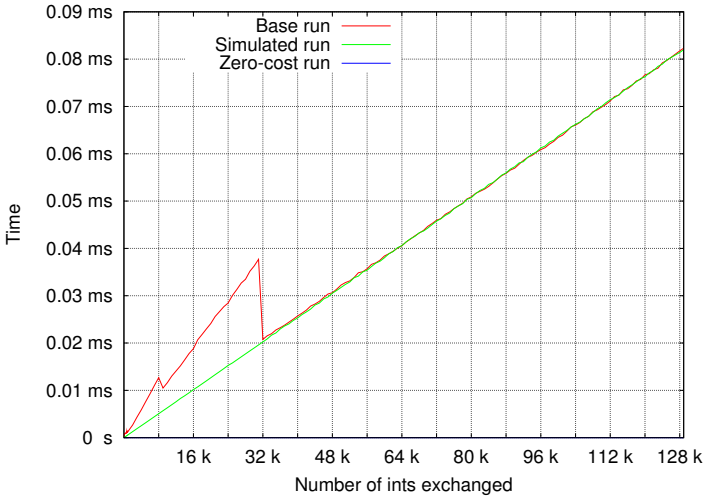

Figure 2. All-to-all on 64 nodes

stand-alone mode. The middle line shows the application when run under the simulator with the optimistic model for collective operations. Finally, the bottom line shows the performance when collectives are free.

The numbers shown are the runtime of the inner loop of Abinit. The application has a fairly large serial part that is not interesting to our experiments and is therefore not shown here.

4.4 Results Summary

The NAS parallel benchmarks are not at all sensitive to the performance of collective operations, while the all-to-all benchmark shows huge performance differences when we vary the cost of collective operations. These are expected results.

There is clearly an impact of collective performance on the runtime of Abinit. When reducing the cost of collective operations to zero, the runtime of the inner loop for our particular atomic simulation is cut in half. However, since Abinit shows such poor scaling, that improvement does not help much.

Table 6 shows the number of messages of a given size sent by Abinit. These numbers are for the 43-atom model run on 48 nodes. No messages larger than 64 kB are sent. 75% of the messages are 4 kB or less, and half of all messages, are 16 bytes (or less).

In Figure 2 we see that the collective model of our simulator is a particularly poor fit for the actual performance of the machine in the range $\leq$ 128 kB (32,000 integers). Since the Abinit run above is in this range, the model (the middle line in Figure 3) is inaccurate for this run. However, since we are only interested in the zero-cost collectives, this does not matter for now.

More work on a better collectives model and more measurements are needed to better characterize the impact collectives have on an application like Abinit. Furthermore,

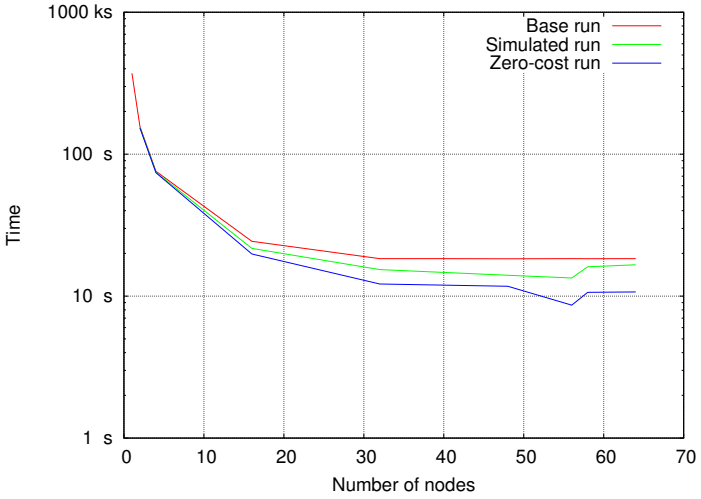

Figure 3. Abinit with 43 atoms

Table 6. Message sizes used by Abinit

Size	≤ 16	≤ 64	≤ 256	≤ 1k	≤ 4k	≤ 16k	≤ 64k
Num	9,263,982	4,371	12,972	51,982	5,378,586	4,085,710	1,361,590
Size	≤ 256k	≤ 1M	≤ 4M	≤ 16M	≤ 64M	≤ 256M	Huge
Num	0	0	0	0	0	0	0

the results shown in this paper may not apply to applications that show better scaling than Abinit.

5 Conclusions and Future Work

We have shown how a hybrid MPI simulator can be used to evaluate the sensitivity of benchmarks and applications to the cost of collective operations. This is interesting because there are many different approaches to improving collective operations. Some of them are difficult to implement and it is not immediately clear, which applications might benefit from these efforts.

We expected that a simple all-to-all benchmark which repeatedly calls **MPI_Alltoall()** inside a tight loop will benefit greatly when we set the cost of collective operations to zero. We also expected that the NAS parallel benchmarks will not gain any benefit from faster collective operations. There was a performance gain for the Abinit application on the problem we ran. Further work with larger input sets and more scalable applications is needed to determine the overall effect of collective operations performance.

6 Acknowledgments

George Riley, Georgia Tech, has helped a lot by teaching the lead author about parallel discrete event simulation and helping shape ideas for the supercomputer simulation project. Many thanks go to Arun Rodriguez for several insightful discussions. We would also like to thank Keith Underwood for suggesting the supercomputer simulation project, and the other team members, Ron Brightwell and Jim Tomkins, for their helpful comments.

References

1. David Bailey, Tim Harris, William Saphir, Rob Van der Wigngaart, Alex Woo, and Maurice Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, December 1995.
2. X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, Ph. Ghosez, J.-Y. Raty, and D.C. First-principles computation of material properties : the ABINIT software project. *Computational Materials Science 25, 478-492*, 2002.
3. X. Gonze, G.-M. Rignanese, M. Verstraete, J.-M. Beuken, Y. Pouillon, R. Caracas, F. Jollet, M. Torrent, G. Zerah, M. Mikami, P. Ghosez, M. Veithen, J.-Y. Raty, and Olevano. A brief introduction to the ABINIT software package. *Z. Kristallogr.*, 220:558, 2005.
4. Torsten Hoefler, Rebecca Janisch, and Wolfgang Rehm. Improving the parallel scaling of abini. In *Accepted to be published in "Science and Supercomputing in Europe"*, 2005.
5. Torsten Hoefler, Rebecca Janisch, and Wolfgang Rehm. Analyzing the parallel scaling of teter's conjugate gradient based minimization for *ab initio* calculations. In *Submitted to the IEEE Cluster 2006 Conference*, 2006.
6. P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864, 1964.
7. Rolf Riesen. **In Submission:** a hybrid MPI simulator. In *IEEE International Conference on Cluster Computing (CLUSTER'06)*, 2006.

8. Michael P. Teter, Micheal C. Payne, and Douglas C. Allan. Solution of Schroedinger's equation for large systems. *Physical Review B*, pages 12255–12263, 1989.