A Comparison of Application Performance Using Open MPI and Cray MPI

Richard L. Graham, Oak Ridge National Laboratory,
George Bosilca, The University of Tennessee, and
Jelena Pješivac-Grbović, The University of Tennessee

Abstract

Open MPI is the result of an active international Open-Source collaboration of Industry, National Laboratories, and Academia. This implementation is becoming the production MPI implementation at many sites, including some of DOE's largest Linux production systems. This paper presents the results of a study comparing the application performance of VH-1, GTC, the Parallel Ocean Program, and S3D on the Cray XT4 at Oak Ridge National Laboratory, with data collected on up to 1024 process runs. The results show that the application performance using Open MPI is comparable to slightly better than that obtained using Cray-MPI, even though platform specific optimizations, beyond a basic port, have yet to be done in Open MPI.

1 Introduction

Since most High Performance Computing (HPC) applications use the Message Passing Interface (MPI) for their inter-processor communications needs, scalability and performance of MPI affect overall application performance. As the number of processors these applications use grows, application scalability and absolute performance is often greatly impacted by that of the underlying MPI implementation. As the number of processors participating in MPI communications increase, the characteristics of the implementation that are most important for overall performance change, and while at small processor count quantities such as small message latency and asymptotic point-to-point message bandwidth are often used to characterize application performance, other factors become more important for determining overall application performance. At large scale, the scalability of the internal MPI message handling infrastructure, as well as the performance of the MPI collective communications operations are key to overall application performance.

This paper present the first study of applications that run on the Cray systems at the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory, comparing overall code performance as a function of the MPI implementation employed. The applications used include Chimera, GTC, and POP, codes of key importance at NCCS. The performance of these applications is studied using two different MPI implementations, Open MPI and Cray-MPI, over a range of processor counts. Because Cray-MPI is a closed source implementation, comparisons are limited to gross performance data, such as overall timing data.

2 Background

2.1 Open MPI

Open MPI [12, 22] is a recent Open Source implementation of both the MPI-1 [20, 24] and MPI-2 [13, 16] standards. Its origins in LA-MPI [2, 14], LAM/MPI [6, 26], FT-MPI [9, 11], and PACX-MPI [18]. The original XT3 port is described in [3]. Open MPI is composed of three major software pieces, the MPI layer (Open MPI), a run-time layer, Open Run-Time Environment (OpenRTE) [7], and the Open Portability Access Layer (OPAL).

OPAL provides basic portability and building block features useful for large scale application development, serial or parallel. A number of useful functions provided only on a handful of platforms (such as asprintf, snprintf, and strncpy) are implemented in a portable fashion, so that the rest of the code can assume they are always available. High resolution / low perturbation timers, atomic memory operations, and memory barriers are implemented for a large number of platforms. The core support

code for the component architecture, which handles loading components at run-time, is also implemented within OPAL. OPAL also provides a rich reference counted object system to simplify memory management, as well as to implement a number of container classes, such as doubly-linked lists, last-in-first-out queues, and memory pool allocators.

OpenRTE provides a resource manager (RMGR) to provide process control, a global data store (known as the GPR), an out-of-band messaging layer (the RML), and a peer discovery system for parallel start-up (SDS). In addition, OpenRTE provides basic datatype support for heterogeneous network support, process naming, and standard I/O forwarding. Each subsystem is implemented through a component framework, allowing whole-sale replacement of a subsystem for a particular platform. On most platforms, the components implementing each subsystem utilize a number of underlying component frameworks to customize OpenRTE for the specific system configuration. For example, the standard RMGR component utilizes additional component frameworks for resource discovery, process start-up and shutdown, and failure monitoring.

A key design feature of this implementation is the extensive use of a component architecture, the Modular Component Architecture (MCA) [25], which is used to achieve Open MPI's poly-morphic behavior. All three layers of the Open MPI implementation make use of this feature, but in this paper we will focus on how this is used at the MPI layer to achieve a portable, and flexible implementation. Specifically, we will focus on the Point-To-Point and Collective-communications implementations.

2.1.1 Point-To-Point Architecture

We will provide only brief description of the Point-To-Point architecture, as a detailed description of the Point-To-Point architecture is given in [15]. The Point-To-Point Management Layer (PML) implements all logic for point-to-point MPI semantics such as standard, buffered, ready, and synchronous communication modes, synchronous and asynchronous communications, and the like. MPI message transfers are scheduled by the PML. Figure 1 provides a cartoon of the three PMLs in active use in the Open MPI code base – OB1, DR, and CM. These PMLs can be grouped into two categories based on the component architecture used to implement these PMLs, with OB1 and DR forming one group, and CM in a group by itself. The following subsections describes these PMLs, as well as the lower level abstractions developed to support these.

As Figure 1 shows, the OB1 and DR PML design is based multiple MCA frameworks. These PMLs differ in the design features of the PML component itself, and share the lower level Byte Transfer Layer (BTL), Byte Management Layer (BML), Memory Pool (MPool), and the Registration Cache (Rcache) frameworks. While these are illustrated and defined as layers, critical send/receive paths bypass the BML, as it is used primarily during initialization and BTL selection. These components are briefly described below.

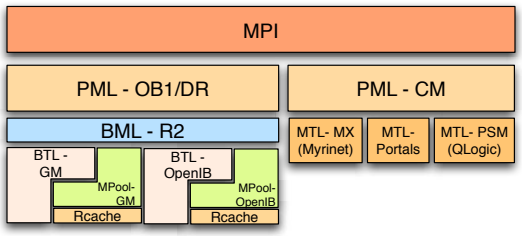

Figure 1: Open MPI's Layered Architecture

The policies these two PMLs implement incorporate BTL specific attributes, such as message fragmentation parameters and nominal network latency and bandwidth parameters, for scheduling MPI messages. These PMLs are designed to provide concurrent support for efficient use of all the networking resources available to a given application run. Short and long message protocols are implemented within the PML, as well as message fragmentation and re-assembly. All control messages (ACK/NACK/MATCH) are also managed by the PML. The DR PML differs from the OB1 PML primarily in that DR is designed to provide high performance scalable point-to-point communications in the context of potential network failures. The benefit of this structure is a separation of the high-level (e.g., MPI) transport protocol from the underlying transport of data across a given interconnect. This significantly reduces both code complexity and code redundancy while enhancing maintainability, and provides a means of building other communications protocols on top of these components.

The common MCA frameworks used in support of the OB1 and DR PMLs are described briefly below.

MPool The memory pool provides memory allocation/deallocation and registration / deregistration services. For example, InfiniBand re-

quires memory to be registered (physical pages present and pinned) before send/receive or RDMA operations can use the memory as a source or target. Separating this functionality from other components allows the MPool to be shared among various layers. For example, MPI_ALLOC_MEM uses these MPools to register memory with available interconnects.

Rcache The registration cache allows memory pools to cache registered memory for later operations. When initialized, MPI message buffers are registered with the Mpool and cached via the Rcache. For example, during an MPI_SEND the source buffer is registered with the memory pool and this registration may be then be cached, depending on the protocol in use. During subsequent MPI_SEND operations the source buffer is checked against the Rcache, and if the registration exists the PML may RDMA the entire buffer in a single operation without incurring the high cost of registration.

BTL The BTL modules expose the underlying semantics of the network interconnect in a consistent form. BTLs expose a set of communication primitives appropriate for both send/receive and RDMA interfaces. The BTL is not aware of any MPI semantics; it simply moves a sequence of bytes (potentially non-contiguous) across the underlying transport. This simplicity enables early adoption of novel network devices and encourages vendor support. There are several BTL modules currently available; including TCP, Myrinet/GM, Myrinet/MX, Cray Portals, Shared Memory (SM), Mellanox VAPI, and OpenIB VAPI.

BML The BML acts as a thin multiplexing layer, allowing the BTLs to be shared among multiple upper layers. Discovery of peer resources is coordinated by the BML and cached for multiple consumers of the BTLs. After resource discovery, the BML layer may be safely bypassed by upper layers for performance. The current BML component is named R2.

The CM PML is designed to provide an MPI interface directly utilizing APIs that expose matching send/receive semantics capable of supporing MPI communication protocols. As the matching logic is implemented in the underlying network library, the CM component is much smaller than the OB1 or DR components. CM handles memory management for requests and buffer management for MPI's buffered sends. The other aspects of MPI's point-to-point semantics are implemented by the Matching Transport Layer (MTL) framework, which provides an interface between the CM PML and underlying network library. Currently there are three implementations of the MTL, for Myricom's MX library, QLogic's InfiniPath library, and the Cray Portals communication stack.

In this paper we will collect application performance data using both the Portals CM and the OB1 PMLs.

2.1.2 Collectives Architecture

The MPI collective communicates in Open MPI are also implemented using the MCA architecture. Of the collective communications algorithm components implemented in Open MPI, the component in broadest use is the Tuned Collectives component [10]. This component implements several versions of each collective operation, and currently all implementations are based on PML level Point-To-Point communications, with run-time selection logic used to pick which version of the collective operations are selected for a given instance of the operation. Selection logic can be determined both at run-time, as well as at compile time. The default algorithm selection is implemented using compile-time decision functions, which select an algorithm based on the communicator and message size parameters of the collective. The run-time algorithm selection is implemented via MCA parameters, and allows user either to specify a particular algorithm for the complete duration of the program run, or to specify a set of rules such that different algorithms can be invoked based on the collective parameters. Figure 2 shows how the MPI collective components fits within Open MPI's component architecture.

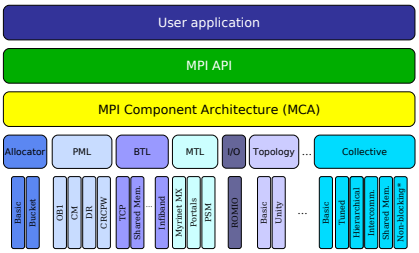

Figure 2: Open MPI's Component Architecture

2.2 Cray-MPI

Cray MPI is derived from MPICH-2 [21], and supports the full MPI-2 standard, with the exception of MPI process spawning. This is the MPI implementation shipped with the Cray Message Passing Toolkit.

2.3 VH-1

VH-1 is a multidimensional ideal compressible hydrodynamics code written in FORTRAN. It is based on the Lagrangian remap version of thePiecewise Parabolic Method (PPM) developed by Paul Woodward and Phil Collela [4].

2.4 GTC

The Gyrokinetic Toroidal Code [19] (GTC) uses first-principles kinetic simulation of the electrostatic ion temperature gradient (ITG) turbulence in a reactor-scale fusion plasma to study turbulent transport in burning plasmas. The simulations aim to improve the understanding of, and to gain the knowledge of controlling the turbulent transport in fusion plasmas for the parameter regimes relevant to magnetic fusion experiments such as the International Thermonuclear Experimental Reactor (ITER).

2.5 Parallel Ocean Program

The Parallel Ocean Program [8] is one of the components of the Community Climate System Model which is used to provide input to the Intergovernmental Panel on Climate Change assessment. POP is the component which models ocean behavior.

2.6 S3D

S3D [17] is a state of the art code, developed at Combustion Research Facility (CRF), Sandia National Laboratory. It is used for Direct Numerical Simulations of turbulent combustion by solving the reactive Navier-Stokes equations on a rectilinear grid.

3 Results

In this section we compare the performance of two MPI implementations on the Cray XT4, Open MPI and Cray-MPI. We compare the results of simple latency and bandwidth measurements, as well as the results of full application runs. In addition, since the performance of many scientific applications is sensitive to the performance of MPI collectives, we also present application performance data using several different collective communication algorithms.

MPI Implementation	Latency
Open MPI - CM	4.91
Open MPI - OB1	6.16
Cray MPI	4.78

Table 1: Zero Byte MPI Latency (usec)

3.1 Experimental Setup

All calculations were run on NCCS's Jaguar cluster. This cluster is made up of a total of 11,508 dual socket 2.6 GHz dual-core AMD Opeteron chips, and the network is a 3-D torus with the Cray-designed SeaStar [1] communication processor and network router is designed to offload network communication from the main processor. The compute nodes run the Catamount lightweight microkernel, allowing for scalable, low-perturbation operations. All communications use the Portals 3.3 communications interface [5].

The default Cray-MPI installation, XT/MPIT version 1.5.31, with default settings are used for the benchmark runs. The trunk version of Open MPI (1.3 pre-release) is used for these runs, with data collected using both the Portal ports of the CM and OB1 PMLs. Open MPI's tuned collectives are used for collective operations. To minimize differences in timings due to processor allocations, all runs for a given application and processor count are run within a single resource allocation.

3.2 Latency and Bandwidth Data

Latency and bandwidth measurements are the measurements often used to assess the quality of an MPI implementation, and as such are included here.

The latency measurements are taken are measured as the half round-trip latency of a zero-byte MPI ping-pong measurement. The latency is measured between process running on two different nodes, not between different cores on the same node. As Table 1 shows Cray-MPI's latency is lower than that of Open MPI's CM, it is 0.13 micro-seconds, or 2.3%, lower than Open MPI's CM latency, and 1.38 micro-seconds, or 29%, lower than Open MPI's OB1 latency.

Bandwidth measurements are taken using the NetPipe [23] benchmark, with the bandwidths measured between two processes running on different nodes. The bandwidth profile in Figure 3 shows that past one-hudred bytes or so, Open MPI's bandwidth is a little higher than that of Cray-MPI's, asymptoting to a similar bandwidth. However, Cray-MPI's bandwidth curve is consistently higher than that of Open MPI's OB1 protocol. It is also noteworthy that Open MPI's transition from the short-message protocol to the long-message protocol is much smoother than that of Cray-MPI's.

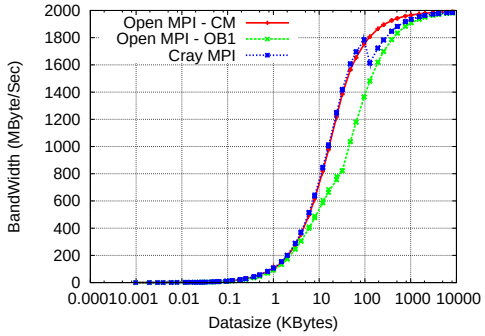

Figure 3: NetPipe Bandwidth Data (MB/sec)

3.3 Application performance Data

In this section we compare the performance of four of the key-codes that use the Cray's XT4 at the NCCS. We study the performance of VH-1, POP, and S3D, over a range of process counts.

The results of the VH-1 performance measurements are presented in Figure 4, for runs in the range of 16 to 256 processors. Overall, Open MPI slightly outperforms Cray-MPI over this set of runs, with the OB1 implementation out-performing Cray-MPI by as much as 4% at 16 processors, and as little as 0.5% at 256 processors. For this particular set of runs, application performance using Open MPI's OB1 point-to-point communications protocol is very similar to that using Open MPI's CM protocol.

The performance of the GTC code is presented in Table 5, for processor counts in the range of four to 1024 processes. In general, for this set of runs, GTC runs faster using Open MPI's OB1 protocol, than when using Open MPI's CM protocol, with a notable exception at 1024 processes, where the CM protocol gives an application times about ten percent lower than us-

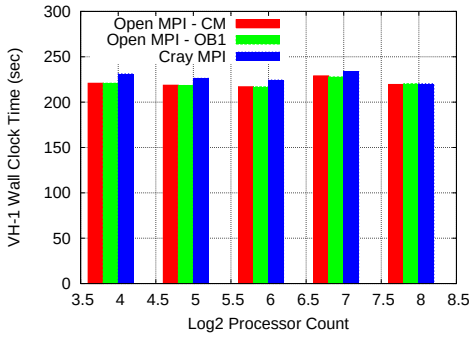

Figure 4: VH-1 Run-Time Data

ing OB1. Over the range of processes, the application runs using Open MPI's CM are faster than the Cray-MPI runs for all but at 128 and 1024 processes. At four processors, the Open MPI OB1 run is about seven percent faster than the Cray-MPI run, and about three percent faster than the Open MPI CM run. At 1024 processes, the Cray-MPI run is about three percent faster than the Open MPI CM run, and about 15% faster than the Open MPI OB1 run.

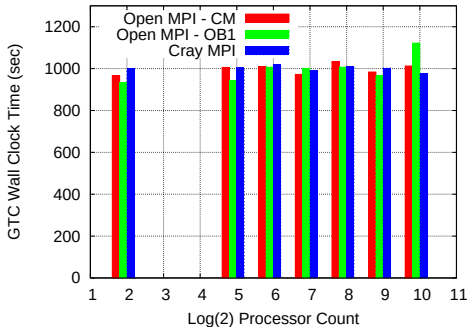

Figure 5: GTC Run-Time Data

Three quantities are used to measure the performance of the POP code, overall time step data, the time spent in the baroclinic phase, where near-neighbor communications are the dominant communications pattern, and the baratropic phase, where an MPI all-reduce is the dominant communications feature. One degree simulations were performed on the range of sixteen to 1024 processes. Due to an Open MPI problem which is being resolved, we were not able to carry out the Open MPI CM runs at 512 and 1024 process counts.

For this set of runs, the data in Table 6 shows

that for this application, the Open MPI CM point-to-point communications algorithm usually outperforms the Open MPI OB1 runs. At sixteen processor count, both benchmark runs ran in about the same amount of time, and at 256 processor count CM outperforms OB1 by about 18%. The Cray-MPI runs slightly outperform the Open MPI CM implementation at smaller process count, but at 128 process count the trend is reversed, with Open MPI CM outperforming Cray-MPI by about three percent at 256 processor count.

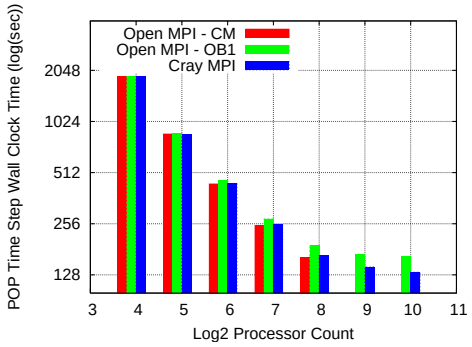

Figure 6: POP Step Run-Time Data

The baroclinic phase timing information is given in Table 7. As mentioned above, this phase is dominated by near neighbor communications. Similar to the total step time timings, Open MPI CM generally outperforms Open MPI OB1. However, Cray-MPI always slightly outperforms Open MPI CM over the entire range of processes counts used in this experiment, and by as much as two percent at the 256 processor count.

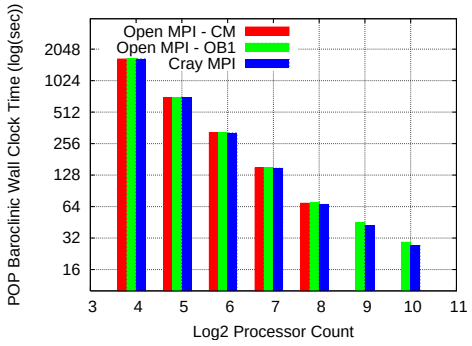

Figure 7: POP Baroclinic Phase Run-Time Data

The Baratropic phase is dominated by MPI all-reduce operations, and does hamper scalability of the POP application, as the number of processes used for this phase increases. From Table 8 we see that of the three implementations used, the Open MPI CM scales best. The time for the Cray-MPI series of runs starts to increase in the range of 128 to 256 processor count, the time for the Open MPI OB1 runs starts to go up in the range of 64 to 128 processors. However, the Open MPI CM runs start have reached their minimum around 256 processors, being about 10% faster than the Cray-MPI run, and about 60% faster than the Open MPI OB1 run.

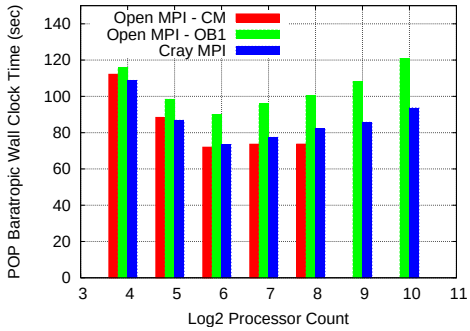

Figure 8: POP Barotropic Phase Run-Time Data

Finally, Figure 9 presents the results of a series of benchmark runs performed with the S3D code. For these runs, the Open MPI CM runs are typically shorter than the Open MPI OB1 runs. At lower process counts - up to 128 - there is no particular performance trend comparing the Open MPI CM timings and the Cray-MPI timings. However, starting at 256 processes, Open MPI CM is consistently performing better than Cray-MPI, and outperforms it by about 12% at 1024 processes.

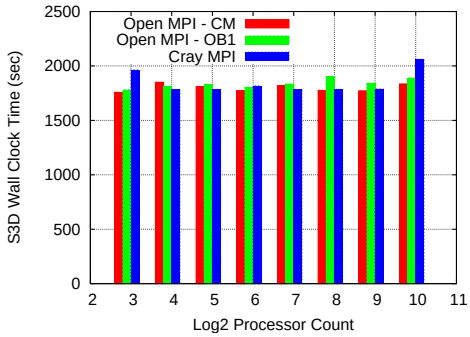

Figure 9: S3D Run-Time Data

3.4 Impact of All-Reduce collective algorithms on Application Performance

Given the importance of the performance of collective operations for overall application performance, we used the four MPI all-reduce algorithms available in Open MPI's Tuned-Collective's collective component to study the performance impact of the particular collective algorithm used on the overall application runtime. Figure 10 presents overall Step time to run the one degree resolution POP problem using 256 processes with Open MPI's CM point-to-point communications algorithm. for reference, the total time to run the same simulation with Cray-MPI is also provided. In the Open MPI runs, the best time is obtained using a binary-tree fan-in reduction, followed by a fan-out broadcast, which is about twelve percent faster than when using either a recursive-doubling or a ring algorithms. As expected, the linear reduce, followed by a linear broadcast performs very poorly. The simulation time for the run using Cray-MPI is within less than one percent of that with Open MPI CM.

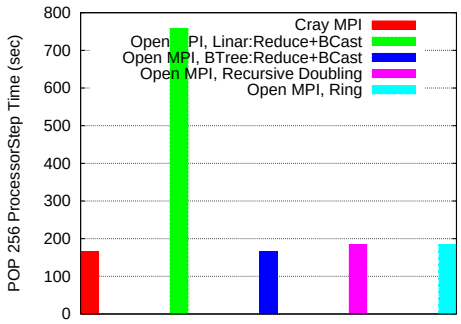

Figure 10: POP 256 Processor Run-Time Data: Impact of All-Reduce Algorithm Employed

4 Discussion

Section 3 presents a large amount of comparative data comparing the performance of Open MPI and Cray-MPI. A detailed analysis of these results is difficult at best, as we don't have access to the Cray-MPI source code, and can only infer from performance characteristics what algorithms might be used.

Open MPI uses two different point-to-point communications schemes, CM and OB1 described in section 2. Aside from some extra latency added by the broad set of communication features that OB1 adds to the cost of point-to-point communications, OB1 and CM have a fundamental difference in how they handle the long message protocol.

The OB1 protocol uses a rendezvous protocol with an eager limit of 32K bytes, on the receive side the memory descriptors are configured to buffer this data of unexpected messages. For large messages, the OB1 protocol attempts to keep network congestion down, so sends only a header used for matching purposes. Once the match is made, the portals get method is used to deliver the users data in a zero copy mode, if the MPI data type is contiguous, directly to the destination. This mode of point-to-point communications is very useful when an application run uses a lot of unexpected messages, i.e. when the message is sent to the destination, before the receive side has posted a matching receive.

The CM protocol is very aggressive on sending data, and for both the short and the long protocol, sends all user data at once. If there is a matching receive posted, the data is delivered directly to the user destination. In the absence of such a posted receive, short messages, i.e. messages shorter than 32K bytes, are buffered by the Portals memory descriptor, but all he data associated with long messages is dropped, and a Portals get request is performed after the match is made to obtain the data. This protocol is aimed at providing the highest bandwidth possible for the application.

Analyzing the simple ping-pong bandwidth curve is figure 3 could lead one to the wrong assumption that when using Open MPI one should always use the CM protocol. The results in section 3 indicate this is not the case, and as the discussion above shows, the best protocol to use really depends on the particular application being run. For POP and S3D, the CM protocol seems to be the better method, but for VH-1 and GTC, OB1 seems to produce lower run-times, at least at lower process counts.

Also, for applications that make use of collective communications, the algorithm of choice can have a large impact on overall performance. As Figure 10 shows, POP performance at 256 processes is very sensitive to the algorithm being used, with the binary-tree based reduction followed by broadcast giving much better performance than the three other algorithms available. The actual algorithm used in Open MPI runs was changed to this one, based on early POP benchmark runs. Open MPI's Tuned-Collectives are such that they provide the flexibility of providing fine grain control over the collectives algorithms being used, with

system wide defaults, as well as with run-time defaults. Further work is being done to improve this support.

Application performance of Open MPI, on average, seems to be comparable to slight better than the same benchmarks run with Cray-MPI. As mentioned above, a discussion of the root causes can not be provided. The advantage of using Open MPI in this context is that as an active Open Source project, with both research and production code development, the performance over Portals improves over time with out any direct work on the Portals specific code, as developers continue to contribute code that improves both point-to-point and collective algorithm performance, as well as other aspects of the code.

5 Conclusions

This paper details a study of the Application performance of several of the key simulation codes that run on the Cray-XT4 system at NCCS in Oak Ridge. The performance of these applications is studied using Open MPI and Cray-MPI, showing that, on average, the the performance using Open MPI is comparable to slightly better than the performance obtained using Cray-MPI, even when the simple latency and bandwidth measurements seem to favor Cray-MPI. Open MPI also has the advantage of having quite a few algorithmic options for point-to-point communications, as well as for collective operations, allowing user, if they so desire, to fine tune the methods used to the needs of their long-time running applications.

As an active Open Source development project, work is currently going on improving the point-to-point performance of Open MPI, as well as that of the MPI collective operations. In particular several efforts are being looked at as to how to implement support for hierarchical collectives, and specifically support for shared memory optimizations, with work from LA-MPI, PACX-MPI, and Sun-MPI **??** being brought to bear on this.

Thanks

This work was supported by a grant from

References

[1] Robert Alverson. Red storm. In *Invited Talk, Hot Chips 15*, 2003.

[2] Rob T. Aulwes, David J. Daniel, Nehal N. Desai, Richard L. Graham, L. Dean Risinger, Mitchel W. Sukalski, Mark A. Taylor, and Timothy S. Woodall. Architecture of LA-MPI, a network-fault-tolerant mpi. In *Los Alamos report LA-UR-03-0939, Proceedings of IPDPS*, 2004.

[3] Brian W. Barrett, Ron Brightwell, Jeffrey M. Squyres, and Andrew Lumsdaine. Implementation of open mpi on the cray xt3. In *46th CUG Conference, CUG Summit 2006*, 2006.

[4] J. M. Blondin and E. A. Lufkin. The piecewise-parabolic method in curvilinear coordinates. *The Astorphysical Journal*, 88:589–594, October 1993.

[5] Ron Brightwell, Tramm Hudson, Arthur B. Maccabe, and Rolf Riesen. The portals 3.0 message passing interface. Technical Report SAND99-2959, Sandia National Laboratories, 1999.

[6] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[7] R. H. Castain, T. S. Woodall, D. J. Daniel, J. M. Squyres, B. Barrett, and G .E. Fagg. The open run-time environment (openrte): A transparent multi-cluster environment for high-performance computing. In *Proceedings, 12th European PVM/MPI Users' Group Meeting*, Sorrento, Italy, September 2005.

[8] J. K. Dukowicz, R.D. Smith, and R.C. Malone. A reformulation and implementation of the bryan-cox-semter ocean model on the connection machine. *J. Atmospheric and Oceanic Tech.*, 10:195–208, 1993.

[9] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. HARNESS and fault tolerant MPI. *Parallel Computing*, 27:1479–1496, 2001.

[10] Graham Fagg, George Bosilca, Jelena Pješivac-Grbović, Thara Angskun, and Jack Dongarra. Tuned: A flexible high performance collective communication component developed for open mpi. In *Proceedings of 6th Austrian-Hungarian workshop on distributed and parallel systems (DAPSYS)*, Innsbruck, Austria, September 2006. Springer-Verlag.

[11] Graham E. Fagg, Edgar Gabriel, Zizhong Chen, Thara Angskun, George Bosilca, Antonin

Bukovski, and Jack J. Dongarra. Fault tolerant communication library and applications for high perofrmance. In *Los Alamos Computer Science Institute Symposium*, Santa Fe, NM, October 27-29 2003.

[12] E. Garbriel et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[13] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, W. Saphir, T. Skjellum, and M. Snir. MPI-2: Extending the Message-Passing Interface. In *Euro-Par '96 Parallel Processing*, pages 128–135. Springer Verlag, 1996.

[14] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant message-passing system for terascale clusters. *International Journal of Parallel Programming*, 31(4), August 2003.

[15] Richard L. Graham, Brian W. Barrett, Galen M. Shipman, Timothy S. Woodall, and George Bosilca. Open mpi: A high performance, flexible implementation of mpi point-to-point communications. *Parallel Processing Letters*, 17(1):79–88, March 2007.

[16] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI — The Complete Reference: Volume 2, the MPI-2 Extensions*. MIT Press, 1998.

[17] E.R. Hawkes, R. Sankaran, J.C Sutherland, and J.H. Chen. Direct numerical simulation of turbulent combustion: Fundamental insights towards predictive models. *Journal of Physics: Conference Series*, 16:65–79, June 2005.

[18] Rainer Keller, Edgar Gabriel, Bettina Krammer, Matthias S. Mueller, and Michael M. Resch. Towards efficient execution of parallel applications on the grid: porting and optimization issues. *International Journal of Grid Computing*, 1(2):133–149, 2003.

[19] Z. Lin, T. S. Hahm, W. W. Lee, W. M. Tang, and R. B. White. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science*, 281:1835, 1998.

[20] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.

[21] Mpich2, argonne. `http://www-unix.mcs.anl.gov/mpi/mpich2/`.

[22] Open mpi. `http://www.open-mpi.org`.

[23] Q.O. Snell, A.R. Mikler, and J.L. Gustafson. In *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[24] Marc Snir, Steve W. Otto, Steve Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, 1996.

[25] Jeffrey M. Squyres and Andrew Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In Vladimir Getov and Thilo Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[26] J.M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, Lecture Notes in Computer Science, Venice, Italy, September 2003. Springer-Verlag.

About the Authors

Richard L. Graham is a staff member at the the National Center for Computational Sciences at Oak Ridge National Laboratory. He received his PhD in Theoretical Chemistry from Texas A&M University. He is an MPI and Tools software lead in the Technology Integration Group, and one of originators of the Open MPI collaboration. He can be reached at Oak Ridge National Laboratory, P.O. Box 2008, Mail Stop 6008, Oak Ridge, TN, 37831-6008. Email: rlgraham@ornl.gov.

Jelena Pjesivac-Grbovic is a Graduate Research Assistant at the Innovative Computing Laboratory at University of Tennessee, working toward a Ph.D. degree in Computer Science. She received a M.S. in Computer Science from University of Tennessee, Knoxville and B.S. degrees in Computer Science and Physics from

Ramapo College of New Jersey. Her research interests are collective communication, parallel communication libraries and computer architectures, scientific and grid computing, and modeling of biophysical systems. She is an active developer on Open MPI project. Address: ICL, Computer Science Department, 1122 Volunteer Blvd. St. 413, Knoxville, TN 37996. Email: pjesa@cs.utk.edu

George Bosilca is a Research Scientist at the Innovative Computing Laboratory at University of Tennessee. He received his Ph.D. degree from Laboratoire de Recherche en Informatique Orsay, Universite Paris XI. His research topics cover High Performance and Distributed Computing. He lead the research and development around several fault tolerant software frameworks and their integration into todays parallel environments and programming paradigms. He was one of the originators of the Open MPI community effort and continue to be one of the active developers. He can be reached at 1122 Volunteer Blvd. Claxton Bldg. Suite 413, Knoxville, TN, 37996. Email: bosilca@cs.utk.edu.