Using Application Communication Characteristics to Drive Dynamic MPI Reconfiguration

Manjunath Gorentla Venkata, Patrick G. Bridges, and Patrick M. Widener
Department of Computer Science
University of New Mexico
Albuquerque NM 87131
$\{$*manjugv,bridges,pmw*$\}$ *@cs.unm.edu*

Abstract

Modern HPC applications, for example adaptive mesh refinement and multi-physics codes, have dynamic communication characteristics which result in poor performance on current MPI implementations. Current MPI implementations do not change transport protocols or allocate resources based on the application characteristics, resulting in degraded application performance. In this paper, we describe PRO-MPI, a Protocol Reconfiguration and Optimization system for MPI that we are developing to meet the needs of dynamic modern HPC applications. PRO-MPI uses profiles of past application communication characteristics to dynamically reconfigure MPI protocol choices. We show that such dynamic reconfiguration can improve the performance of important MPI applications significantly when exact communication profiles are known. We also present preliminary data showing that profiles from past application runs with different (but related) inputs can be used to optimize the performance of later application runs.

1. Introduction

Emerging HPC applications such as adaptive mesh refinement (AMR) and multi-physics codes have dynamic communication characteristics that pose challenges for modern message passing libraries. For example, both the number of communicating processes and the size of messages can vary widely over the course of a single run in applications built using SAMRAI, a MPI adaptive mesh library [1]. HPC applications such as HyperCLaw [2], Sweep3D, and Sphot [1] also exhibit similar communication characteristics.

Current MPI implementations are not designed to handle dynamic changes in application communication behavior. Modern MPI implementations, for example, choose between either one-sided (RDMA) or two-sided (Send-Receive) communication protocols, depending on the number of communication peers each process is expected to have. When the chosen protocol does not match the communication characteristics of an application, performance suffers: either memory management and polling overheads increase, or network round-trip latencies do. Furthermore, if the number of peers varies widely, as it does in the applications mentioned above, any static protocol choice will cause these performance issues for at least a portion of the application's execution.

To address these problems, we are developing Protocol Reconfiguration and Optimization for MPI (PRO-MPI), a system for improving the performance of parallel applications with dynamic communication characteristics. PRO-MPI uses profiles of past application communication behavior as a basis for dynamic reconfigurations of message passing functionality to better suit expected application communication requirements. Our current prototype focuses on dynamically changing which connections use one-sided and two-sided communication protocols over the course of an application run, reducing both polling overheads and network protocol overheads. We demonstrate that such protocol reconfiguration can improve the performance of important AMR applications, and present preliminary data demonstrating that communication profiles collected from one application run can be used to improve the performance of later application runs with different (but related) inputs.

The rest of the paper is structured as follows. Section 2 provides basic background on the Open MPI MPI implementation on which PRO-MPI is currently implemented, as well as the high-performance message passing strategies that PRO-MPI reconfigures between to improve application performance. Section 3 presents the design of our system, its prototype implementation in Open MPI, and an example reconfiguration we have implemented. Section 4 presents experimental results demonstrating PRO-MPI's ability to improve application performance by adapting to changing application communication characteristics, and Section 5 discusses related work. Finally, Section 6 presents directions for future work and concludes.

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

2. Background

2.1. Open MPI

Open MPI is a MPI implementation that uses a well-defined component architecture, the MPI Component Architecture (MCA) [3]. Communication functionality in MCA is provided by self-contained software modules that support well-defined interfaces. For example, MCA defines interfaces for implementing different collective communication algorithms and supporting different network interfaces, allowing Open MPI to be customized to run on a wide range of platforms.

Open MPI uses the modules at the PML (point-to-point messaging layer) and BTL (byte-transfer layer) to support point-to-point communication over specific networking hardware. The BTL provides an interface for moving bytes over a given communication fabric, for example shared memory, Ethernet, or Infiniband. The PML provides higher-level services such as message fragmentation and reassembly and matching necessary to support the MPI interface over a given BTL layer. Our work has focused primarily on optimizing the the use of OpenIB BTL, a BTL module that transfers data over Infiniband links using OpenIB's verbs interface.

2.2. Infiniband Communication Protocols

Infiniband supports two different protocols for small-message transfer:

1) A two-sided *send-recv* protocol in which the receiver posts a set of receives and waits for notification that any of them are complete by polling a single event queue, and
2) A one-sided *RDMA* protocol in which each sender has direct remote access to a memory buffer in which to place messages, and the receiver is notified by polling receive buffers.

RDMA protocols allow for lower network protocol latencies, but polling costs can overwhelm these savings when many RDMA connections are used.

3. PRO-MPI Design and Implementation

3.1. Design and Prototype Implementation

To support profile-based protocol reconfiguration and optimization, PRO-MPI supplements standard MPI implementations with three additional elements. A *profiler* collects information about application communication characteristics into a communication profile for later use; an *analyzer* uses globally collected profiling information to generate schedules for driving reconfiguration in later application runs; and

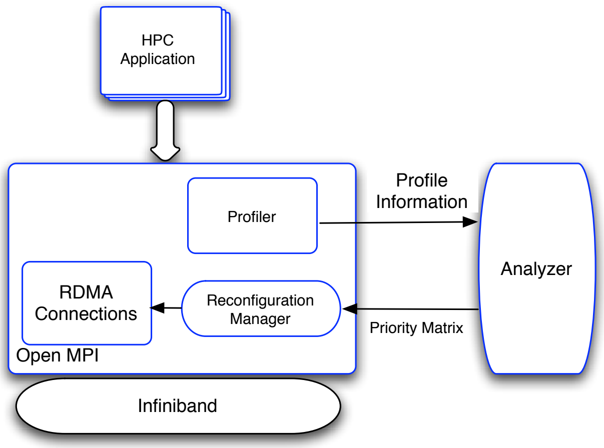


Figure 1: PRO-MPI components and their use to control a prototype protocol reconfiguration

a *reconfiguration manager* dynamically changes communication settings based on previously generated schedules[1].

We have implemented a prototype version of PRO-MPI, shown in Figure 1, as an extension to the Open MPI MPI implementation. In this implementation, the runtime PRO-MPI elements, the profiler and reconfiguration manager, are part of Open MPI framework that interact with existing layers to implement profiling and reconfiguration. At this point, we have focused on using them to reconfigure communication at the Open MPI BTL layer, as described below; our implementation, however, is designed to allow other reconfigurations, for example of Open MPI collective communication modules.

3.2. Example Reconfiguration

Using PRO-MPI, we have prototyped a protocol reconfiguration designed to improve the performance of applications that dynamically change how many and with which peers they communicate frequently at runtime. Specifically, we have implemented connection-protocol adaptation between one-sided and two-sided communication protocols at the Open MPI BTL layer based on profiles gathered from previous application runs. In this reconfiguration, the PRO-MPI profiler gathers information about how often a process communicates with each peer during application-indicated phases. We also added a one line call to each application to indicate to PRO-MPI when a new application phase begins (phases could also potentially be inferred from application behavior, for example calls to `MPI_barrier`).

1. While the profiler, analyzer, and reconfiguration manager could potentially interact at runtime without the need for off-line static profile analysis, this potentially adds global monitoring, analysis, and synchronization overheads that may be difficult to recover.

In subsequent runs on related (but not necessarily identical) inputs, the reconfiguration manager controls which BTL communication protocol is used to communicate with each peer in each application phase using a schedule generated from the profiling information. This allows PRO-MPI to dynamically control which peers use RDMA channels, improving overall application performance as described in Section 4.

The profiling information and analysis needed for this reconfiguration is relatively simple. Specifically, the profiler gathers the number of small messages sent to each peer by each process during each indicated application phase. The analyzer uses the profile information to generate a *priority matrix* as well as the maximum number of RDMA connections allowed for each process ($maxRDMA$). The 3-dimensional priority matrix contains the communication priority between any two MPI processes during each application phase; specifically element $p[i][j][k]$ of the priority matrix contains the priority of communication from process i to j at application phase k.

The reconfiguration manager initiates, resets and manages RDMA connections based on data in the priority matrix generated by the analyzer from profiling information. At a given application time step, two MPI processes p (the initiator) and q may be connected via RDMA if the priority of that connection is higher than other MPI processes, and the current number of RDMA connections p is less than $maxRDMA$. Figure 2 shows how connections could be reconfigured between between MPI process i and other MPI processes over an application run when $maxRDMA = 2$.

The reconfiguration manager also controls how the communication protocol of a MPI process is switched between RDMA and Send-Receive. At each reconfiguration point, all RDMA connections change state from CONNECTED to QUIESCE. In the QUIESCE state, RDMA connections cannot be used for any new message transfers. After all pending messages are transferred, the resources allocated to the connection are freed and the RDMA connection state is changed to DISCONNECTED. The connection state for a process is changed back to CONNECTED only after the process establishes a new RDMA connection to another MPI process.

4. Evaluation

4.1. Overview

To demonstrate the capabilities of PRO-MPI we ran experiments to measure the impact of profile-based protocol reconfiguration on both synthetic microbenchmarks and application performance. Our experimental platform is a 22-node SGI Altix ICE system with an Infiniband interconnect. Each compute node has 2 3.0GHz Intel Xeon processors, 16GB of memory, and runs SUSE Linux with kernel version 2.6.16 and OpenFabric's OpenIB network stack.

We tested PRO-MPI using synthetic communication benchmark and several application frameworks. The synthetic benchmark can dynamically change which peers it communicates with, how many message it exchanges with each peer, and how big each messages is. The application frameworks we tested were:

- **SAMRAI**, a popular C++ software framework developed to implement parallel adaptive multi-physics applications. In our experiments, we ran SAMRAI's spherical shock wave problem for 10 time steps with the coarsest domain fixed to $[40, 40, 40]$ and varying refinement levels.
- **HyperCLaw**, a hybrid C++/Fortran AMR code developed and maintained by Lawrence Berkeley National Laboratory. Our experiments used an AMR gas dynamics application data set which models the interaction of a Mach 1.25 shock in air hitting a helium bubble for 10 time steps with base grid of size $[32, 8, 8]$.

Each test was run in three modes: using native Open MPI, using PRO-MPI to collect communication profile information under low mesh refinement conditions (3 for SAMRAI, 2 for HyperCLaw), and using that collected profile information to drive PRO-MPI's reconfiguration decisions; reconfiguration decisions are made at reconfigurable points which, in these tests, are application time steps. We also used the low-refinement profiles to drive reconfiguration for a set of high mesh refinement runs (4 for SAMRAI, 3 for HyperCLaw).

4.2. Synthetic Benchmark Results

To understand and estimate the performance benefits of using connection-protocol reconfiguration, we implemented a synthetic benchmark that imitates changing communication characteristics of an application, and then compared its performance while using profiled PRO-MPI and Open MPI. In these experiments, the benchmark used a profile that exactly characterized the changing communication characteristics to drive PRO-MPI 's reconfiguration so as to measure the best-case performance improvement that PRO-MPI could expect.

Figure 3(a) shows the impact of connection-protocol reconfiguration on the benchmark's runtime as the number of processes participating in the communication increases, while Figure 3(b) shows the impact of reconfiguration when the number of messages sent by each process increases. As expected, protocol reconfiguraion increases in effectiveness with both increasing number of processes and messages sent per process, with 40% or greater benchmark performance improvements over OpenMPI's static choice of connections to send-receive or RDMA protocol.

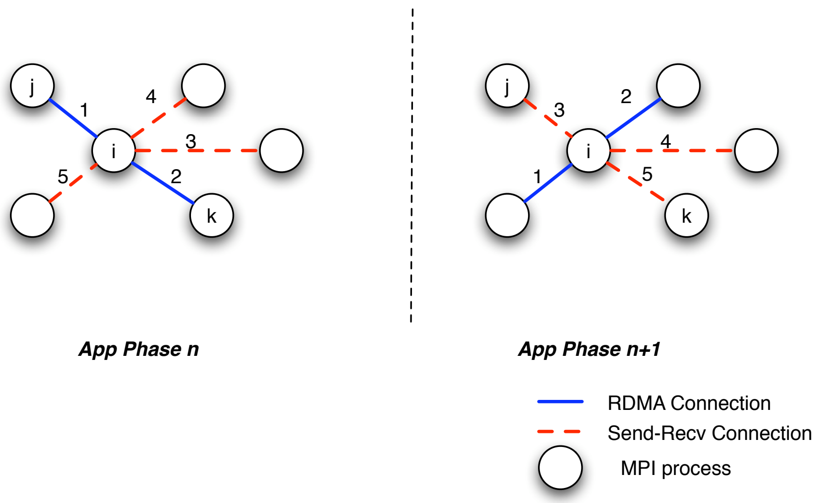
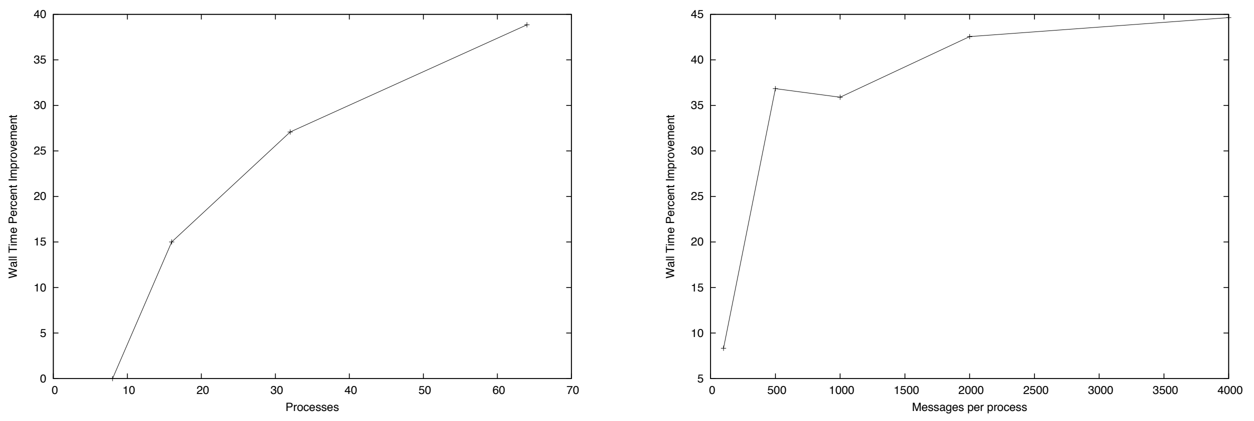


Figure 2: Example connection reconfiguration between RDMA and *Send-Receive* connections. The nodes in the figure represent MPI processes, while the number on the edges represent the communication priority of the processes during an application phase.



(a) Increasing number of processes where each process sends 1000 msgs of size 1kB

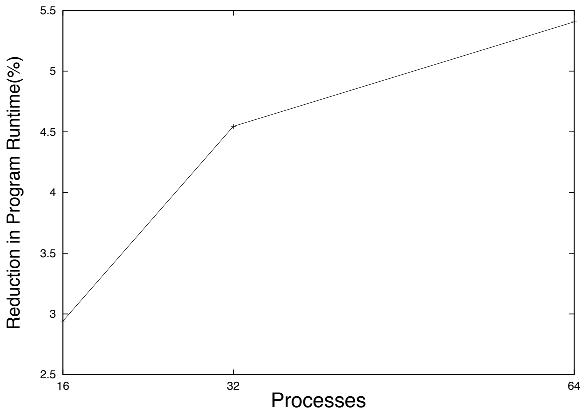
(b) Increasing messages sent per process in a 32-process problem

Figure 3: PRO-MPI performance improvement over Open MPI on synthetic communication benchmark

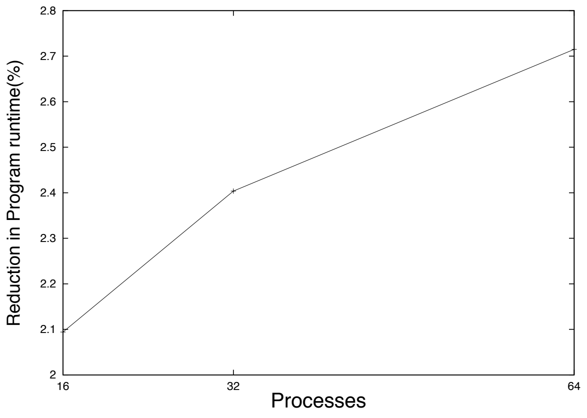
4.3. Application Performance with Exact Profiles

To measure PRO-MPI's potential protocol reconfiguration impact on SAMRAI and HyperCLaw applications at different node counts, we compared the performance of Open MPI runs and profiled PRO-MPI runs of these applications at low refinement levels on 16, 32, and 64 nodes. Low mesh refinement levels were used both to limit application runtime and because runtimes from high refinement runs have high standard deviation in our experiments. Each test was run 5 times.

Figure 4 shows the performance impact of reconfiguration on SAMRAI and HyperCLaw problems with exact performance profiles. SAMRAI performance is improved by 5.5% for a 64 process problem, and the performance improvement increases as the number of processes in the problem increases. Similarly, HyperCLaw performance improves by 2.7% for the 64 process problem, and the performance improvement increases slowly with increasing process counts. We also observe from table 1 that performance improvement increases with different input data sets, specifically, increasing the refinement.



(a) SAMRAI performance improvement



(b) HyperCLaw performance improvement

Figure 4: Performance improvement using PRO-MPI with exact profiles.

Testcase	Runtime (sec)		%Diff.
	Open MPI	PRO-MPI	
HyperCLaw	846±50	716±43	16±10
SAMRAI	391±1	387±1	2

Table 1: Performance improvement using profile collected at low mesh refinement (3 for SAMRAI, 2 for HyperCLaw) to optimize communication in application run with higher mesh refinement (4 for SAMRAI, 3 for HyperCLaw) on 32 nodes. Numbers represent the average of 5 runs.

4.4. Application Performance with Inexact Profiles

Because exact profiles may not be available or may be difficult or time-consuming to produce, we have also done preliminary work studying how profiles from one run can be used to improve the performance of an application on related input data.

Table 1 shows that performance profiles collected from lower-refinement runs can be used to substantially improve the performance of more time-consuming high-refinement runs. Specifically, profiles from refinement 2 HyperCLaw runs can be used to improve the performance of refinement 3 HyperCLaw runs by approximately 16%; we also observed performance improvement in SAMRAI runs, though more modest ones than those observed in HyperCLaw. Finally, we note that the standard deviation of the preliminary HyperCLaw numbers is high.

4.5. Analysis

To understand the source of the performance observed improvements we profiled the communication characteristics of the applications while they used PRO-MPI and Open MPI for communication. On 64 nodes, HyperCLaw sends up to 20% RDMA messages when using PRO-MPI and up to 1% when using Open MPI with the same number of allowed RDMA connections. Similarly, SAMRAI sends 13 times more RDMA messages when using PRO-MPI instead of Open MPI with the same number of RDMA connections. This demonstrates that PRO-MPI's performance improvements are the result of more efficient use of RDMA connections compared to random assignment of RDMA connections to peers.

Although using PRO-MPI for communication has performance benefits, it adds overhead of creating and destroying RDMA connections at every reconfiguration point. Also, using RDMA requires dedicated memory buffers and a certain amount of polling overhead. Our results show that these overheads can be amortized when sufficient messages are exchanged between processes during each application phase. Also, applications can realize significant performance benefits while still using a relatively low $maxRDMA$ value in order to limit both memory usage and polling overhead. For our tests, we used $maxRDMA = 4$; attempting to use more RDMA connections (e.g. $maxRDMA = 8$) did not improve application performance in tests we ran.

5. Related Work

The use of profiling, reconfiguration, and adaptation to improve application performance has been an area of active research. Adaptive MPI [4] and Maghraoui [5] both used adaptation and reconfiguration to improve load balancing for MPI applications, for example, while we have shown in previous work [6] how reconfiguration can be used to improve bandwidth availability. Other systems have shown that profile data can be used for optimizing MPI performance at link time or launch, STAR-MPI [7] for selecting appropriate collective communication algorithms and HP-MPI [8] for placing MPI processes in a system. In contrast, our work uses profile data for fine-grained runtime reconfiguration and also provides a framework that can be used to implement other similar reconfigurations.

There have also been several efforts which use dynamic connection management to improve application performance. Yu [9] uses adaptive connection management to create RDMA connections on demand, but created connections in this system are never reset or torn down if they become idle; Shipman [10] describes a similar connection management strategy for Open MPI. In applications with dynamic communication characteristics, both of these approaches would use significantly more resources than our approach, as every created RDMA connection occupies memory and would need to be polled for message availability.

6. Conclusions and Future Work

In this paper, we show that using the communication characteristics of HPC applications to drive reconfiguration can improve the performance of those applications. Our PRO-MPI framework collects profile information about the communication characteristics of MPI-based applications and uses those profiles to drive connection-protocol reconfiguration. We have demonstrated that PRO-MPI improves performance both in an application built using the SAMRAI adaptive-mesh library and the HyperCLaw application. Our results also show that PRO-MPI can improve performance even in cases when profiles derived from application runs with different input sets are used to drive reconfiguration.

We think it will be increasingly important to consider the dynamic characteristics of emerging HPC applications while designing HPC system software. To this end, we first intend to study the impact of profile-based reconfiguration, using various input sets and problem sizes, on SAMRAI-based codes, HyperCLaw and other similar HPC applications. Second, we will work to identify architecture and design areas of current MPI implementations that will present performance challenges for emerging HPC applications. We will also improve the design and implementation of PRO-MPI and enhance its support for additional reconfigurations.

Acknowledgement

The authors sincerely thank HPC@UNM for providing hardware to run the experiments. We also thank Andy Wissink for his help in configuring SAMRAI framework. Lastly, we thank the US Department of Energy's Office of Science for their financial support to this work.

References

[1] Jeffrey S. Vetter and Frank Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. *J. Parallel Distrib. Comput.*, 63(9):853–865, 2003.

[2] Shoaib Kamil, Ali Pinar, Daniel Gunter, Michael Lijewski, Leonid Oliker, and John Shalf. Reconfigurable hybrid interconnection for static and dynamic scientific applications. In *CF '07: Proceedings of the 4th international conference on Computing frontiers*, pages 183–194, New York, NY, USA, 2007. ACM.

[3] 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.

[4] Chao Huang, Gengbin Zheng, Laxmikant Kalé, and Sameer Kumar. Performance evaluation of adaptive MPI. In *PPoPP '06: Proceedings of the Eleventh ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pages 12–21, New York, NY, USA, 2006. ACM.

[5] Kaoutar El Maghraoui, Boleslaw K. Szymanski, and Carlos Varela. An architecture for reconfigurable iterative MPI applications in dynamic environments. In *Proc. of the Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM2005), number 3911 in LNCS*, pages 258–271. Springer Verlag, 2005.

[6] Manjunath Gorentla Venkata and Patrick G. Bridges. MPI/CTP: A reconfigurable MPI for HPC applications. In Dieter Kranzlmüller, Peter Kacsuk, Jack Dongarra, and Jens Volkert, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 13th European PVM/MPI Users' Group Meeting*, volume 4192 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[7] Ahmad Faraj, Xin Yuan, and David Lowenthal. STAR-MPI: self tuned adaptive routines for MPI collective operations. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 199–208, New York, NY, USA, 2006. ACM.

[8] David G. Solt. A profile-based approach for topology aware MPI rank placement. In *High Performance Computation Conference (HPCC), Springer Lecture Notes in Computer Sciences*, 2007.

[9] Weikuan Yu, Qi Gao, and D.K. Panda. Adaptive connection management for scalable MPI over InfiniBand. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, April 2006.

[10] Galen M. Shipman, Timothy S. Woodall, Richard L. Graham, Arthur B. Maccabe, and Patrick G. Bridges. Infiniband scalability in OpenMPI. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.