MPI_Reduce: Introducing OpenMPI

JEFF SQUYRES

I'm setting aside my usual monthly *MPI Mechanic* monthly column to write a full-length article about a new Message Passing Interface (MPI) implementation: *Open MPI*. Open MPI effectively represents the merger, or dare I say "reduction", of three previously separate MPI implementations: LAM/MPI from Indiana University (of which I am the lead developer), LA-MPI from Los Alamos National Laboratory, and FT-MPI from the University of Tennessee. Over the course of the past year, The Ohio State University and the University of Stuttgart have also been added to the Open MPI team. *Table One* shows the current list of Open MPI members and their prior work.

The driving goals of the Open MPI project are as follows:

- Write a maintainable, open source, production-quality MPI implementation
- Emphasize high performance on a wide variety of platforms
- Implement all of MPI-1 and MPI-2 (including true support for multi-threaded MPI applications and asynchronous progress)
- Pool our collective MPI implementation expertise and eliminate replicated effort between multiple MPI projects
- Take only the best ideas from our prior projects
- Provide a flexible MPI implementation suitable for a wide variety of run-time environments and parallel hardware
- Create a platform that enables world-class parallel computing research
- Enable the greater HPC community to contribute

The first public release of Open MPI will be a beta-quality version available at the SC conference in Pittsburgh in November 2004. An abbreviated list of features is listed below:

- Support endian and network heterogeneity (striping message passing across multiple networks)
- Natively support TCP/IP, shared memory, Myrinet, Quadrics, and Infiniband networks
- Support a wide variety of run-time environments
- Inherently based on a "pluggable" component architecture, allowing MPI applications to choose their back-end networks and run-time environments at execution time
- Open source with a BSD-like license

How Did This Happen?

The Open MPI project started as a result of a chance meeting at a conference in late 2003 between members of the LAM/MPI, LA-MPI, and FT-MPI projects. Over the next two months, casual conversations between team members turned into formal teleconferences that yielded a surprising insight (to us, at least): although each of our three implementations each had their own strengths and weaknesses, we shared a lot of functional/code overlap.

Specifically, there were major portions of code that we had each written that performed essentially the same tasks in more or less the same way. In hindsight, this duplication should not have been much of a revelation — each of the three projects were MPI implementations, after all — but it was an eye-opening experience for us.

The obvious implication was that we should collaborate in order to reduce efforts replicated between our projects. At SC2003, over a series of breakfast-through-lunch meetings, we decided to write an entirely new MPI implementation that, although strongly influenced by our existing projects, would be a whole new code base. The main idea was to take all the best ideas from our current projects and put them into the new implementation. This new effort seemed to be a perfect op-

portunity to "take the best; leave the rest": extract the best work from each project and jettison old kruft that had built up over the years. Starting from scratch gave us the perfect opportunity to learn from past mistakes and "do it right this time."

And that's exactly what we did.

Over the past 12 months, Open MPI has grown into [almost] a full MPI-2 implementation that not only represents the superset of the previous projects, but also includes new designs and functionality not previously available in other MPI implementations. The rest of this article will provide an overview of the features and research available in Open MPI, as well as outline our future plans and directions.

What's the Point?

The most frequently asked question that we get is: "Why did you do this?" The most obvious answer is to reduce replicated efforts between our previously separate projects. By uniting under one MPI implementation, we have pooled resources such as: developers, testers, software engineering know-how, MPI implementation experience, and testbed platforms. For example: why continue to debug TCP socket-level in each of our three previous projects? Having a single implementation of TCP socket code frees up developers to work on other aspects of Open MPI.

Hence, not only did the total amount of work *decrease*, the total number of developers *increased* (at its largest, the Open MPI project had 24 developers actively writing code). Having so many experienced MPI developers has allowed us to make massive progress in a relatively short amount of time. By completing the majority of an MPI-2 implementation in 12 months, we can now focus on more research-oriented projects — using Open MPI as the vehicle for that research.

Although message passing is a relatively well-understood paradigm, there are still many unanswered questions that will require investigation and rigorous research.

We plan to explore them with Open MPI. Indeed, we have already had the chance to do so with our design and implementation of the MPI point-to-point communication. For example, Open MPI supports true multi-threaded MPI applications while striping large messages across heterogeneous networks.

We are not alone in this desire — there are many third parties who wish to conduct research in parallel computing as well. Open MPI was designed and constructed on a component-based architecture. As such, Open MPI is a collection of small, independent components that are composed at run-time to form an MPI implementation. Third parties (e.g., vendors and researchers) can write and components and distribute them independently of Open MPI. The learning curve for each of the component types was deliberately designed to be low, making the barrier to entry as small as possible.

Open MPI also represents our latest work in high-performance point-to-point message passing. Preliminary results indicate that Open MPI has ping-pong latencies comparable to other MPI implementations but with bandwidths up to 30 percent higher in high-speed TCP networks. See the paper "Open MPI's TEG point-to-point communications methodology: Comparison to existing implementations" on the Open MPI webite at *www.open-mpi.com*.

Finally, we have effectively created a superset of our previous work — all the best features and cutting-edge research are (or soon will be) included in Open MPI. Since Open MPI is a production-quality product, life should be simpler for the end user.

In the first release of Open MPI,

At its largest, the Open MPI project had 24 developers actively writing code

TABLE ONE

Open MPI Member Organizations

ORGANIZATION	PRIOR WORK
The Advanced Computing Laboratory (CCS-1), Los Alamos National Laboratory	LA-MPI
High Performance Computing Center Stuttgart (HLRS), University of Stuttgart	PAC-X MPI
Innovative Computing Laboratory, Department of Computer Science, University of Tennessee	FT-MPI
Network Based Computing Laboratory, Department of Computer Science and Engineering, The Ohio State University	MVAPICH
Open Systems Laboratory, Pervasive Technologies Lab at Indiana University	LAM/MPI

many popular networking types (TCP/IP, shared memory, Infiniband, Quadrics, and Myrinet) and several back-end run-time systems (rsh/ssh, RMS, and BProc) are natively supported — each are affected through standalone components. This feature gives users the ability to run Open MPI in a wide variety of parallel environments.

Support for more networks and run-time systems will be added in the near-term after the SC04 release.

Open MPI Features

Some of the most notable features of Open MPI are listed below. Individually, some of the features are not "new" (e.g., MPI implementations that support Infiniband have been around for years). However, the fact that they are all combined under a single MPI implementation is not only new, it is genuinely useful to end users, system administrators, and vendors (i.e., having a single MPI implementation installation that simultaneously supports Infiniband, Quadrics, and Myrinet networking).

Full MPI-2 Implementation

A full MPI-2 implementation is something that open source MPI projects have been chasing for years. Previous MPI implementations have been based on monolithic software architectures that — regardless of how well-abstracted and logically constructed — are highly complex software packages, presenting a steep learning curve for new developers and third parties.

These existing code bases typically locked development into highly specific implementation models and were simply not designed with flexibility or extensibility in mind. This restriction effectively prevented extensions that did not already conform to existing models (such as MPI-2 one-sided operations and dynamic processes).

Each of our own prior MPI implementations suffered from this problem, which is one of the motivating reasons to start a new MPI implementation. We therefore designed Open MPI from the ground up to be both flexible and capable of all MPI-2 concepts. The SC04 release of Open MPI will not include *all* of MPI-2, but it will be close (e.g., there simply wasn't time to implement the MPI-2 one-sided operations before November).

Support Common Network Types

The majority of clusters use some shared memory, Infiniband, Quadrics, Myrinet networking, and/or some form of TCP/IP (e.g., Gigabit Ethernet). Open MPI natively supports all of these network types. Specifically, an application that is compiled with Open MPI can utilize any of these networks without recompiling or re-linking.

The Message Passing Interface (MPI)

The Message Passing Interface (MPI) is the *de facto* standard for message passing parallel programming on large-scale distributed systems. One of the main goals of the MPI standard is to enable portability: parallel applications that run on small, development platforms (e.g., a small Linux cluster) will also run on larger, production systems (e.g., large clusters or "big iron" specialized parallel hardware).

At the risk of repeating myself from an earlier *MPI Mechanic* column, I typically emphasize the following points when explaining what MPI is:

1. MPI stands for the Message Passing Interface.
2. MPI is a standard defined by a large committee of experts from industry and academia.
3. The design of MPI was heavily influenced by decades of "best practices" in parallel computing.
4. Although typically collectively referred to as the "MPI standard," there are actually two documents (MPI-1 and MPI-2).
5. Implementations of the MPI standard provide message passing (and related) services for parallel applications.
6. There are many implementations of the MPI standard.

The MPI standard defines a set of functions that can be used by applications to pass messages from one MPI process to another. MPI actually defines a lot more services than just message passing — but the heart and soul of MPI is (as its name implies) passing messages between MPI processes.

As noted in point three, there are actually two documents that comprise the MPI standard: MPI-1 and MPI-2. MPI-1 is the "core" set of MPI services for message passing. It provides abstractions and mechanisms for basic message passing between MPI processes (as well as some additional features that are helpful for general parallel computing). MPI-2 is a set of extensions and functionality beyond what is defined in MPI-1 such as dynamic process control, one-sided message passing parallel I/O, etc.

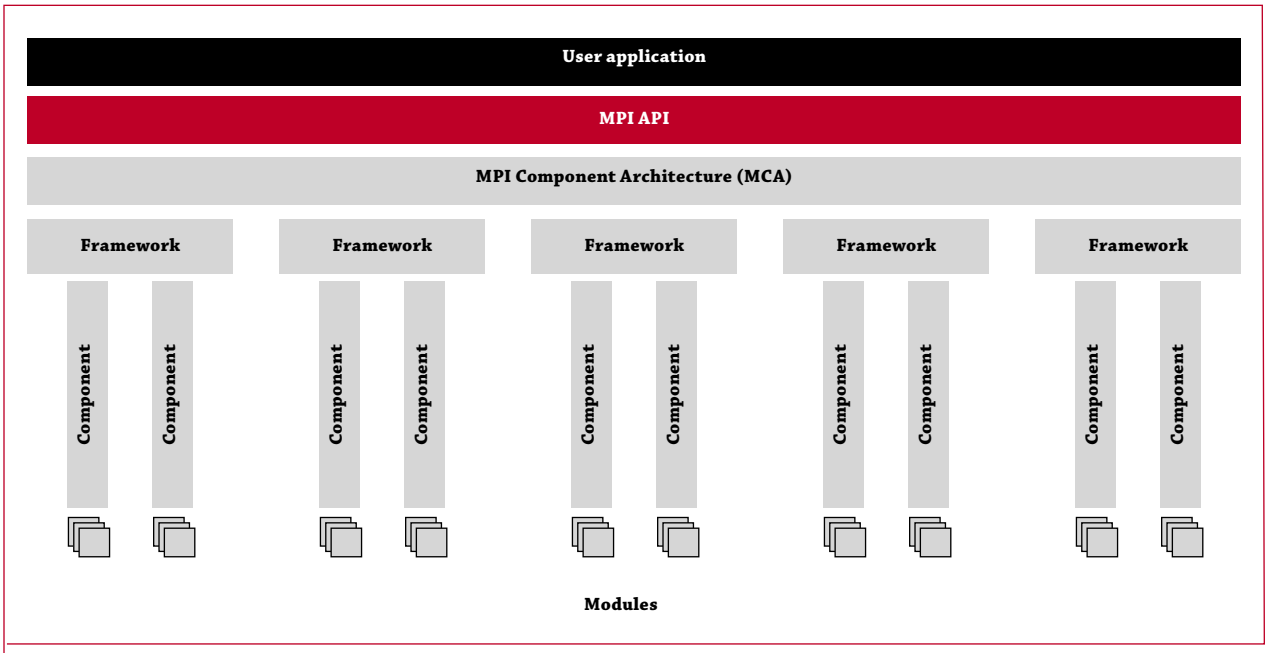

FIGURE ONE: Organization of the MCA

Additionally, Open MPI can use all of these networks *simultaneously* — striping large messages across multiple networks where possible. Detection of which network to use for a given message is largely automatic. For example, if a message is sent from one MPI process to another on the same node, shared memory will be used. If a message is sent to a different node, the "best" available network will be used (i.e., one of the high-bandwidth / low latency networks such as Infiniband, Quadrics, or Myrinet).

Multi-Threaded Applications and Asynchronous Progress

One of the more elusive features of MPI-2 is `MPI_THREAD_MULTIPLE` — the ability to support multiple, concurrent application threads within an MPI library while allowing them all to make simultaneous progress. Open MPI was fundamentally designed to require only a small number of fine-grained locks and local data storage to enable efficient multi-threaded concurrency (the locks are not used in single-threaded situations). True asynchronous progress is also supported, using hidden "progression" threads within the MPI library. While there is typically a latency penalty for asynchronous (due to added thread context switching), most applications structured to exploit overlap of communication and computation will still exhibit a net performance gain.

Support Common Runtime Systems

Many clusters still use `rsh` or `ssh` to start parallel MPI applications. However, a growing number now use other back-end run-time systems such as SLURM, Sun Grid Engine, BProc, PBS, etc. Open MPI expanded on work from LAM/MPI to allow launching of MPI applications in a wide variety of back-end run-time environments. Open MPI can therefore deduce upon execution which run-time environment it is executing under and invoke the appropriate handlers (analogous to how Open MPI automatically figures out which network to use for message communication). `rsh / ssh`, BProc, and RMS clusters will be supported in Open MPI's first release; support for additional run-time environments will be added in the near term after SC.

Component Architecture: A Vehicle for Research

Open MPI is fundamentally based on the MPI Component Architecture (MCA). The MCA is a collection of component frameworks that provide services to Open MPI at run-time. Each framework supports a single API for its services; different implementations of this API are called *components*. When a component is paired with resources, it is called a module. For example, a process running on a compute node that contains two Gigabit Ethernet cards may have two modules of the TCP/IP component in the point-to-point transfer framework. *Figure One* shows how the MCA is conceptually organized. Note that the layers pictured in *Figure One* do *not* represent call stacks at run-time — in most cases, once initialized properly (usually during `MPI_INIT`) modules are invoked directly by the MPI layer for maximum performance.

The following is an abbreviated list of the MPI layer component frameworks in Open MPI, and what they are used for:

- *coll*: MPI collective algorithms. Provide back-end implementations for `MPI_BARRIER`, `MPI_BCAST`, etc.
- *io*: MPI-2 I/O functionality. Currently only supports the ROMIO MPI-2 IO implementation from Argonne National Labs.
- *one*: MPI-2 one-sided operations. Provide back-end implementations for `MPI_GET`, `MPI_PUT`, etc. This framework will not be included in Open MPI's first release.
- *op*: Collective reduction operations. Provide optimized implementations of the intrinsic MPI reduction operations, such as `MPI_SUM`, `MPI_PROD`, etc. This framework will not be included in Open MPI's first release.
- *pml*: MPI point-to-point management layer. This framework is the top layer in point-to-point communications; the PML is responsible for fragmenting messages, scheduling fragments to PTL modules, high level flow control, and reassembling fragments that arrive from PTL modules.
- *ptl*: MPI point-to-point transport layer. This framework is the bottom layer in point-to-point communications; the PTL is responsible for communicating across a specific device or network protocol (e.g., TCP, shared memory, Elan4, GM 1/2.x, Open IB, etc.).
- *topo*: MPI topology support. Provide back-end implementations of all the topology creation and mapping functions.

The PML and PTL are described in detail in the paper "TEG: A high-performance, scalable, multi-network point-to-point communications methodology" on the Open MPI website.s

Components are implemented as shared libraries. Hence, using components means searching directories and loading dynamic libraries (all of which is transparently handled by the MCA). Extending Open MPI's functionality is therefore simply a matter of placing components in the directories that Open MPI searches at run-time. For example, adding support for a new network type entails writing a new PTL component and placing its resulting shared library in Open MPI's component directory. MPI applications will instantly "see" the component and be able to use it at run-time.

Thus, the MCA enables two main actions:

- Run-time decisions about which components to be used. For example, if running an MPI application on an Infiniband network, the "ib" PTL component will automatically be found, selected, and used. Similarly, if MPI processes are on the same node, the "sm" (shared memory) PTL component will automatically be used to communicate between them. As such, MPI applications are independent of the components that comprise Open MPI: components can be added or removed from the system without recompiling or re-linking user applications. This feature is important for software vendors; their product can be shipped without knowledge of what components will be used in a customer's target environment.
- Third parties can develop and distribute their own components. Since components can be added to an Open MPI implementation at any time, components can be distributed independently of the main Open MPI software package. Vendors, therefore, can write and distribute components that maximize performance on their platforms. Developers can actively research parallel computing topics in a production-quality MPI without the steep learning curve required for a monolithic MPI implementation. This concept is explored further in the paper "The component architecture of Open MPI: Enabling third-party collective algorithms," listed in the resources section of this article.

Enable Community Contributions

There are many organizations with a vested interest in a high-quality MPI implementation because much of today's parallel computing is done with MPI. HPC vendors, for example, need to provide a production-quality MPI implementation quickly for new platforms in order to attract users in the HPC community. Users need to have highly tuned MPI implementations that enable their applications to extract maximum performance from their parallel environments. Researchers need a production-quality MPI implementation that allows them to quickly and easily experiment with new methods, techniques, and algorithms.

Indeed, rather that write a new MPI implementation from scratch, most of these needs can be accommodated by writing one or more components that "plug in" to Open MPI. As such, third parties can even distribute components themselves; there is no need to be

included in the main Open MPI distribution nor be tied to Open MPI's release schedule. We want to encourage both kinds of contributions: third-party components as well as attracting vendors and other committed developers join the Open MPI project (see the Open MPI web site for more details).

> Researchers need a production-quality MPI implementation that allows them to quickly and easily experiment with new methods, techniques, and algorithms.

To that end, Open MPI is the first project of Open HPC, Inc., a non-profit organization recently established for the promotion, dissemination, and use of open source software in high-performance computing. Open MPI therefore receives administrative support from Open HPC. Open HPC is somewhat analogous to the Apache Foundation; it is a parent organization that aims to provide administrative support for high-quality open source HPC projects. More information about Open HPC is available at their web site.

Wrapup

With our collective MPI experience, we have rapidly produced a new MPI implementation that incorporates all the best features from several previous projects as well as complete redesigns in some areas traditionally unexplored by MPI implementations. Open MPI focuses on performance, production-quality software, and usability with the intrinsic attitude that it should "just work." Although no software is perfect, we believe that Open MPI is greater than the sum of its predecessors.

A natural question at this point is: "What about FT-MPI, LAM/MPI, LA-MPI, MVAPICH, and PAC-X MPI?" Only time will tell, but it is likely that each project will eventually end when funding stops or be used exclusively as a research vehicle for goals different than Open MPI's. Indeed, some of the projects *must* continue to exist until their current funding expires.

We'd love to get users and vendors involved in the effort. We invite you to visit our web site, *www.open-mpi.org*, download the beta software, and join the announcements and/or general users' mailing list.

Jeff Squyres is a post-doctoral research associate at Indiana University and is the one of the lead technical architects of the Open MPI project. You can reach him at jsquyres@open-mpi.org.

Resources

MPI Forum (including the MPI-1 and MPI-2 specification documents): *www.mpi-forum.org*

MPI — The Complete Reference: Volume 1, The MPI Core (2nd ed) (The MIT Press) by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. ISBN 0-262-69215-5.

MPI — The Complete Reference: Volume 2, The MPI Extensions (The MIT Press) by William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. ISBN 0-262-57123-4.

NCSA MPI tutorial
- *webct.ncsa.uiuc.edu:8900/public/MPI*

The Open MPI Project
- *www.open-mpi.org*

Open HPC, Inc.
- *www.open-hpc.org*

FT-MPI
- *icl.cs.utk.edu/ftmpi*

LA-MPI
- *public.lanl.gov/lampi*

LAM/MPI
- *www.lam-mpi.org*

MVAPICH
- *nowlab.cis.ohio-state.edu/projects/mpi-iba*

PAC-X MPI
- *www.hlrs.de/organization/pds/projects/pacx-mpi*

Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy 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.

Jeffrey M. Squyres and Andrew Lumsdaine. *The component architecture of Open MPI: Enabling third-party collective algorithms*. In Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications, St. Malo, France, July 2004.