High-performance computing (HPC) clusters have a hierarchical architecture. Within this architecture, middleware and libraries bind compute nodes together to perform computations and communicate across nodes. The challenge lies in how to best distribute application processes across the clusters for maximum performance.

One way to address this problem is by using Message Passing Interface (MPI), the de facto communication standard for distributed, shared-memory parallel programming. MPI helps provide application scalability and portability across multiple platforms and operating systems—a critical requirement for HPC clusters. Although MPI middleware and libraries are available commercially, open source versions can be particularly useful for HPC because the source code is available for development, research, and performance tuning. In addition, open source product development is supported by the efforts of many universities, national laboratories, and commercial companies, fostering a community approach to code enhancement.

The Open MPI Project is an example of one such open source project. It receives contributions from national laboratories such as Sandia National Laboratories, Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Oak Ridge National Laboratory, as well as contributions from multiple universities and research institutes and backing from companies such as Cisco Systems, Mellanox Technologies, Voltaire, and QLogic.

Open MPI is an open source implementation of both the MPI-1 specification and the MPI-2 extension to MPI-1. The Open MPI charter calls for a stable, production-quality software protocol that can support multiple high-speed, low-latency interconnects across multiple platforms and architectures. It originated as the merger of three major MPI implementations—Fault-Tolerant MPI (FT-MPI), Los Alamos MPI (LA-MPI), and Local Area Multicomputer MPI (LAM/MPI)—and later incorporated Parallel Computer Extension MPI (PACX-MPI) as well.

When communicating across nodes, each of these MPI implementations has its own strengths and weaknesses at the application programming interface (API) level. For example, FT-MPI and LA-MPI provide process- or network-level fault tolerance: when a network or a node crashes, these implementations can handle the API-level communication and either migrate or re-spawn the job as necessary. Open MPI helps balance these strengths and weaknesses by combining the code, best practices, methodologies, and features of these four MPI standards into a single distribution. This article outlines the Open MPI architecture, describes how to set different runtime configuration options, and offers tips that can help maximize bandwidth performance.
offers tips that can help maximize bandwidth performance.

Understanding Open MPI

Open MPI is network agnostic and includes multi-network and multi-device support. It is designed to support any interconnect, which can be automatically chosen based on selection policies or manually specified by administrators.

As of version 1.1, Open MPI includes comprehensive support for the MPI-1 and MPI-2 standards, including one-sided communications, which were not available in previous versions. Open MPI currently supports common HPC cluster interconnects such as Ethernet, Myricom Myrinet GM and Myrinet Express (MX), and OpenIB and Mellanox VAPI (mVAPI) InfiniBand. It also supports loop-back and shared-memory communication modes. Supported schedulers include Sun N1 Grid Engine, Tera-Scale Open Source Resource and Queue Manager (TORQUE), and LoadLeveler. Platform Load Sharing Facility (LSF) is not supported natively, but can be integrated with Open MPI using separate scripts available from Platform Computing. Open MPI can also use Remote Shell (RSH) or Secure Shell (SSH) for MPI communication.

Open MPI architecture

The Open MPI code is based on a component architecture composed of plug-ins that load at runtime depending on the available networks. The advantage of this architecture emerges during compilation: because the application is not linked directly to the network libraries, it can typically run on any network it finds in the cluster environment. This design helps ease the addition of new network interconnects.

Open MPI includes three primary layers:

- **Open MPI API (OMPI)**: Contains the MPI APIs and supporting logic
- **Open Run-Time Environment (ORTE)**: Provides the interface to OMPI, including the runtime environment
- **Open Portable Access Layer (OPAL)**: Connects directly to the OS

These layers are abstracted from one another, with separate code bases that are compiled into the libmpi, liborte, and libopal libraries, and are classified based on dependency order. Each layer can communicate with the OS directly (see Figure 1)—and because the OMPI code, for example, does not need to go through the two underlying layers to communicate with the OS, this design is particularly advantageous for performance-sensitive function calls.

Modular Component Architecture

Open MPI follows a component-based design, where the components are independent units that can be developed separately and may be added to or removed from the code base without disturbing other components. This approach helps increase flexibility and simplify component development and reuse.

The core of Open MPI is the Modular Component Architecture (MCA). Unlike many component-based architectures, this architecture has a lightweight design, which helps reduce the overhead for operations like component loading and unloading. It consists of three parts, which are assembled at runtime (see Figure 2):

- **Frameworks**: Public interfaces to the MCA that use MCA services for handling components
at runtime; frameworks also own their own internal services

- **Components**: Individual pieces of code that can be plugged into the framework during compilation or runtime
- **Modules**: Instances of MCA components that have the same properties as those components, but also have their own private state

### Setting runtime configuration options

The Open MPI MCA enables runtime configuration of applications compiled with Open MPI. Many of the settings available in other MPI implementations have constant values, reducing the configuration options available at runtime. Open MPI, however, makes these settings available as MCA parameters that can be set at runtime.

For example, using the `-mca` option with the `mpirun` command specifies the framework and component list settings to be used at runtime:

```
mpirun -mca framework component_list
```

In Open MPI, the Byte Transfer Layer (BTL) framework defines the characteristics of MPI communication between compute nodes. By defining the BTL at runtime, administrators can select a particular interconnect for the application and set additional options such as loopback and shared-memory communications among processes.

Several options are available to include or exclude components in a given run. For example, the following command includes only the `openib`, `self`, and `sm` components:

```
mpirun options -mca btl ^openib,self,sm executable
```

The `^` symbol applies to the entire list of MCA parameters given at the command line.

**Note**: Because `mpirun` and `mpiexec` are identical in Open MPI, `mpiexec` can be used in the same format to start MPI applications.

### Alternatives to the command line

Other than the command line, two low-precedence methods are available for setting MCA parameters. One method is to use shell environment variables, using the format `OMPI_MCA_parameter`. For example, the following command has the same effect as the first `mpirun` example in the preceding section:

```
OMPI_MCA_btl=openib,self,sm export OMPI_MCA_btl mpirun options executable
```

The other method uses text files: administrators can add values to either `$HOME/.Open MPI/.openmp_mca` or `/etc/Open MPI-mca-mca-params.conf`, which are then searched in order. The following lines from the example configuration file enable the same settings as the first `mpirun` example in the preceding section:

```bash
#sample mca settings
btl = openib,self,sm
```

Additional component libraries can be provided by adding files in `$HOME/.Open MPI/components`. These components should be made visible to all nodes by sharing the file system or installing the components separately on each node.

Framework details on specific Open MPI installations are available through the `ompi_info` command. Adding the appropriate `--param` option to this command shows the parameters of a particular component. For example, the command `ompi_info --param btl openib` lists the MCA parameters for the `openib` BTL component.

### Maximizing bandwidth performance

A key feature of Open MPI is its adherence to the “law of least astonishment,” which states that when two interface actions conflict, the correct choice should be the outcome that a user or programmer would find the most logical and least surprising. This principle comes into play, for example, when using a high-speed interconnect such as InfiniBand or Myrinet in addition to Ethernet. If the BTL component for the high-speed interconnect is not specified at runtime, Open MPI can automatically sense this and disable the default `tcp` BTL component, allowing MPI communication to take place over the high-speed interconnect.

When using Remote Direct Memory Access (RDMA)–based interconnects like InfiniBand, Open MPI can also preserve the most recently used registrations, which helps increase the

---

“A key feature of Open MPI is its adherence to the ’law of least astonishment,’ which states that when two interface actions conflict, the correct choice should be the outcome that a user or programmer would find the most logical and least surprising.”
bandwidth performance of applications that reuse send and receive buffers. For example, host channel adapters (HCAs) usually send and receive directly from RAM without involving the processor. However, the OS can change the virtual or physical RAM mapping at any time, potentially causing bad messages if the OS changes the physical mapping after an HCA starts sending a message.

Administrators can preserve registrations by setting the MCA parameter `mpi_leave_pinned` to 1. For example:

```bash
mpirun options -mca mpi_leave_pinned 1 executable
```

This command sets the OS to not change the physical mapping until the HCA is finished sending a message. The disadvantage of using this parameter is that because the OS can only support a limited amount of registered memory at one time, the application may run out of registered memory. Therefore, administrators typically should not use this approach except when using latency- and bandwidth-sensitive applications.

Figure 3 shows the positive effect that the `mpi_leave_pinned` parameter can have on bidirectional bandwidth. These results are based on tests run by Dell engineers in June 2007 using the Ohio State University MPI-level bidirectional bandwidth test¹ on two Dell™ PowerEdge™ 1950 servers, each with two quad-core Intel® Xeon® E5320 processors at 1.86 GHz and connected through InfiniBand double data rate (DDR) host bus adapters. At small message sizes, the benchmark returned similar results regardless of whether this parameter was enabled. At large message sizes, however, enabling it resulted in significantly increased bandwidth.

**Utilizing flexible MPI middleware for HPC clusters**

As an open source project, Open MPI benefits from years of research invested in it by organizations in both academia and the high-tech industry. It has several advantages over prior MPI implementations, including one-sided communications, a component-based architecture, and support for multiple high-speed interconnects across a variety of platforms. The inherent flexibility and configurable runtime options of Open MPI make it well suited for many different HPC cluster environments.

**Toby Sebastian** is a senior engineering analyst in the Enterprise Solutions Group at the Dell Bangalore Development Center. Toby has a B.Tech. in Computer Science and Engineering from the University of Calicut. His current interests include HPC clustering packages, high-end interconnects, and performance analysis of parallel applications.

**Sanjay Lalwani** is a senior engineering analyst in the Enterprise Solutions Group at the Dell Bangalore Development Center. Sanjay has a B.E. in Computer Science and Engineering from Bhilai Institute of Technology, Durg, and a postgraduate diploma in Information Technology from the Indian Institute of Technology, Kharagpur. His current interests include parallel applications, cluster file systems, and storage technologies.

**Munira Hussain** is a systems engineer and adviser in the High-Performance Computing Group at Dell. Munira has a bachelor’s degree in Electrical Engineering from the University of Illinois at Urbana-Champaign. She specializes in HPC systems and architecture design. Her areas of interest include high-speed, low-latency interconnect networks such as InfiniBand as well as application tuning and benchmarking.

¹ For more information on this benchmark, see mvgich.cse.ohio-state.edu/benchmarks.