NICHOLAS T. KARONIS

MPICH-G2: An MPI for Grids

In January 1993, researchers from more than 40 organizations from research laboratories, academia, and industry formed the Message Passing Interface Forum. The intention was to define a set of library interfaces, with the goal of producing a widely used standard for message-passing programs. In June 1994, the forum published version 1.0 of the Message Passing Interface (MPI) standard.

As the standard was being developed, members of the forum at Argonne National Laboratory and Mississippi State University began implementing MPI in a library they called MPICH. Early on, they decided to internally split their library into two parts: a "top" part including everything found in the standard (e.g., all the MPI functions application programs call) and a "bottom" part comprising a much smaller set of functions primarily responsible for transporting data. Functions in the top part perform all the parameter checking and implement some MPI functions in terms of others (e.g., `MPI_Bcast` implemented in terms of `MPI_Send` and `MPI_Recv`). Functions in the bottom part focus exclusively on point-to-point communication. To define the interaction between the top and bottom parts of the library, the MPICH developers created the Abstract Device Interface (ADI).

The two-layer design and the new ADI provided a framework in which the standard could be implemented quickly. Industrial developers (HP, SGI, and Intel, to name just a few) were able to produce their own MPI libraries by first acquiring the top portion of the MPICH library, freely distributed by the MPICH team soon after the standard was published, and then concentrating their efforts on the bottom portion, writing only those functions defined in the ADI. This approach allowed vendors to write significantly fewer functions compared to all those defined in the MPI standard. The MPICH design also allowed vendors to ignore all the mundane but necessary tasks required when implementing a library that is directly callable from application programs, and instead focus their efforts on issues they had proprietary expertise in, namely, the efficient movement of data on their systems. The high-performance computing landscape was quickly populated with MPI implementations available on many commercial platforms, and MPI applications soon followed. This early MPICH design decision, along with the concurrent development of the library, proved crucial to the successful adoption of MPI as a widely used standard for message-passing programs.

Why a Grid MPI?

Initially all MPI applications were executed on a single computational resource (e.g., an SMP or cluster), and for many applications a single resource is sufficient. There are, however, MPI applications for which a single computer is not sufficient. Such applications can be broadly described by two categories.

In the first category are applications whose memory or computational requirements exceed the capacity of even the largest and most powerful computers. Many scientific applications fall into this category. Such applications often simulate real-world phenomena (e.g., weather, air flow over helicopter rotors, or even the collision of black holes) by mapping three-dimensional real-world space to a computer's memory. For example, an application that simulates and predicts weather may select a region on a map and carve that space (which includes some vertical distance into the atmosphere) into cubes. Each cube stores all the parameters of interest to the weather model (e.g., temperature, humidity, and wind velocity). The number of parameters within each cube and the total memory available on the computer dictate how many cubes can fit on the computer. That number of cubes, along with the size of the region to be studied, determines the amount of three-dimensional real-world space that each cube represents. Predicting weather for a given region on a computer with a modest amount of memory may result in each cube representing 100x100x100 miles of real-world space. Predicting weather for the same region on a computer with a large amount of memory may result in the same real-world volume being decomposed into many more cubes, with each cube representing less real-world space, say only 10x10x10 miles. The smaller the physical space each cube represents, the greater the accuracy of the prediction. Hence, scientists using these applications are always in search of computers with more memory.

In the second category are applications that require resources not found at any single site. Consider the National Science Foundation TeraGrid, an integrated Grid of computing resources from academic institutions across the United States and interconnected via a dedicated high-bandwidth optical network. One TeraGrid site has a large data storage capacity (San Diego Supercomputing Center, or SDSC), another has a large computing capacity (National Center for Supercomputing Applications, or NCSA), and a third has rare state-of-the-art visualization equipment (Argonne). In Grids like these with separate but integrated resources, applications can form *functional pipelines*. That is, a TeraGrid application can run across these three sites by initially processing the data at SDSC, sending a selected subset of the data to NCSA for more intensive processing that produces visualization data, and then sending that data to Argonne for rendering. By their very nature, the applications in this category require more than one computer.

MPICH-G2: An MPI for Grids

Running an MPI application across multiple computers — each likely to involve different administrative domains and heterogeneous architectures and networks — presents numerous challenges. There are issues of security, process management, scheduling co-allocation, and data conversion, to name a few. To address these challenges, we developed MPICH-G2, a Grid-enabled MPI based on the MPICH library from Argonne and integrated with the Globus Toolkit™ for Grid management.

The Globus Toolkit is a collection of software components designed to support the development of applications for rids. MPICH-G2 uses Globus Toolkit services to support efficient and transparent execution in heterogeneous Grid environments.

The user starts by describing the resources (e.g., computers), requirements (e.g., number of CPUs), and other parameters (e.g., location of application and command arguments) using the *Resource Specification Language* (RSL) to write an *RSL script*. Then, before application startup, the user employs the *Grid Security Infrastructure* (GSI) to obtain a (public key) proxy credential to authenticate to each remote site. This step provides a single sign-on capability.

Once authenticated, the user uses the standard `mpirun` command to launch the application. MPICH-G2 uses the *Dynamically-Updated Request Online Coallocator* (DUROC), a co-allocation library, to schedule and start the application across the various computers named in the RSL script. The DUROC library itself uses the *Grid Resource Allocation and Management* (GRAM) API and protocol to start and subsequently manage a set of subcomputations, one for each computer. For each subcomputation, DUROC generates a GRAM request to a remote GRAM server, which authenticates the user, performs local authorization, and then interacts with the local scheduler to initiate the computation. DUROC holds all the processes at a barrier (hidden in MPICH-G2's `MPI_Init`) until all the subcomputations have started executing, and then DUROC and MPICH-G2 tie the various subcomputations together into a single MPI computation.

GRAM will, if directed in the RSL script, use *Globus Access to Secondary Storage* (GASS) to stage executable(s) from remote locations (indicated by URLs). GASS is also used to direct standard output and error (`stdout` and `stderr`) streams to the user's terminal. Once the application has started, MPICH-G2 selects the most efficient communication method possible between any two processes, using vendor-supplied MPI if available or *Globus communication* (Globus IO) with *Globus Data Conversion* (Globus DC) otherwise.

Other Grid-Related Enhancements

The functionality described above, in some sense, represents the minimum that might be reasonably expected of any Grid MPI — the ability to securely start an application across multiple computers, giving the appearance that the application is running on a single computer by automatically selecting the most efficient point-to-point communication method and converting data between different computer architectures (i.e., big endian vs. little endian) where necessary. In addition to this basic functionality, MPICH-G2 provides other Grid-related enhancements.

After the processes have started, MPICH-G2 uses information specified in the RSL script to create multilevel clustering of the processes based on the underlying network topology. The processes are first grouped based on the sites which they are running, and then subgroups are formed within these groups based on the machines on which they are running. As an illustration, consider an application running on the TeraGrid, with some of the processes running on the SDSC cluster and the remaining processes distributed across the two Argonne clusters. MPICH-G2 would partition the processes into two groups, one for SDSC and the other for Argonne. It would then partition the processes in the Argonne group into two subgroups, one for each of the Argonne clusters. This grouping is done for two reasons. First, some applications find it helpful to query MPICH-G2 to discover this process topology and then use that information to form communicators. Second, and perhaps more important, MPICH-G2 uses this grouping information in its implementation of MPI's collective operations (e.g., `MPI_Bcast`). Performance increases significantly when those operations are implemented in a Grid network topology-aware manner.

By default, MPICH-G2 uses the *Transmission Control Protocol* (TCP) for intercluster messaging. However, we and others have seen that TCP does not perform efficiently on optical wide-area networks. The reason is that the high bandwidth and high latency sometimes found on these networks force TCP to send data in extremely large buffers in order to achieve the full bandwidth utilization. TCP starts with a modest buffer size and quickly increases the buffer (exponentially) to the optimal size. The difficulty with TCP arises when it encounters what it interprets to be a "congestion event" (e.g., a dropped packet). TCP automatically falls into "congestion avoidance" mode and thereafter takes far too long to increase the buffer (only linearly in congestion avoidance mode) to reach its optimal buffer size.

To help alleviate this problem, MPICH-G2 uses the GridFTP protocol from the Globus Toolkit. GridFTP opens *multiple TCP streams* between a pair of processes. Data is carved into pieces on the sending side; these pieces are sent in parallel down the many TCP streams and are reassembled on the receiving side. MPICH-G2 applications can disable TCP and "turn on" GridFTP between any pair of processes, and doing so has been shown to provide increased bandwidth utilization when compared to TCP for large messages sent over networks with a high bandwidth capacity and high latency.

Experiences with Grid MPI Applications

Many groups throughout the world have used MPICH-G2. Here we mention a few experiences representing the two application categories previously described.

MPICH-G2 has been used to form a functional pipeline between computers at Argonne, NCSA, and Amsterdam, using the NetherLight network and then forming another pipeline on the TeraGrid. On the NetherLight network researchers in Amsterdam viewed images by remotely controlling computations at Argonne and NCSA, processing data that was produced by a Rayleigh-Taylor simulation at the ASC/Alliances Center for Astrophysical Thermonuclear Flashes (Flash Center). On the TeraGrid, data stored at SDSC from simulations of Type 1a supernovae also done at the Flash Center was sent to NCSA for processing, with results then sent to Argonne for rendering.

MPICH-G2 has also been used by conventional (i.e., non-pipelining) applications in search of more memory and computing power. Such applications include MM5 climate modeling on the TeraGrid, forming multivariate geographic clusters to produce maps of regions of ecological similarity, studying distributed execution of a large computational electromagnetics code, and exploring automatic partitioning techniques as applied to finite element codes.

MPICH-G2 was awarded a 2001 Gordon Bell Award for its role in an astrophysics application involving the evolution of gravitational waves from colliding black holes. The winning team used MPICH-G2 to run across four supercomputers in California and Illinois, achieving scaling of 88 percent (1,140 CPUs) and 63 percent (1,500 CPUs), computing a problem size five times larger than any other previous run involving Einstein's general relativity equations.

Future Work

Future work on MPICH-G2 will be influenced dramatically by two independent and major research and development efforts. First, the Globus Alliance is about to release version 4.0 of the Globus Toolkit. This new toolkit will be based on *Web services* and represents a fundamentally different approach to providing Grid services. Second, the MPICH team is completing the latest version of the MPICH library, which will include the MPI-2 standard. In extending the library the MPICH team took the opportunity to re-evaluate the original MPICH design and decided to develop a completely new implementation of the MPICH library, including the MPI-1 functions, and to devise a new ADI interface.

Accordingly, MPICH-G2 will have to be modified to respond to the changes in the Globus Toolkit and new MPICH to re-achieve its MPI-1 functionality. In addition, MPICH-G2 will have to be extended to include the MPI-2 standard.

Resources

MPICH-G2
- www.globus.org/mpi

The Globus Toolkit
- www.globus.org

MPICH
- www.mcs.anl.gov/mpi/mpich

MPI Forum
- www.mpi-forum.org

NetherLight
- www.surfnet.nl/innovatie/netherlight

ASC/Alliances Center for Astrophysical Thermonuclear Flashes
- www.flash.uchicago.edu

Nicholas T. Karonis is an associate professor of computer science at Northern Illinois University and a resident associate guest in the Mathematics and Computer Science Division at Argonne National Laboratory. He can be reached at karonis@niu.edu.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced

Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38 with the University of Chicago, and in part by the National Science Foundation.