MPI-2: Extending the Message-Passing Interface

Al Geist[1] and William Gropp[2] and Steve Huss-Lederman[3] and Andrew Lumsdaine[4] and Ewing Lusk[2] and William Saphir[5] and Tony Skjellum[6] and Marc Snir[7]

[1] Oak Ridge National Laboratory
[2] Argonne National Laboratory
[3] University of Wisconsin
[4] University of Notre Dame
[5] NASA Ames
[6] Mississippi State University
[7] IBM Research Yorktown

Abstract. This paper describes current activities of the MPI-2 Forum. The MPI-2 Forum is a group of parallel computer vendors, library writers, and application specialists working together to define a set of extensions to MPI (Message Passing Interface). MPI was defined by the same process and now has many implementations, both vendor-proprietary and publicly available, for a wide variety of parallel computing environments. In this paper we present the salient aspects of the evolving MPI-2 document as it now stands. We discuss proposed extensions and enhancements to MPI in the areas of dynamic process management, one-sided operations, collective operations, new language binding, real-time computing, external interfaces, and miscellaneous topics.

1 Introduction

During 1993 and 1994, a group of parallel computer vendors, library writers, and application scientists met regularly to define a standard interface for message-passing libraries. The result of this effort was MPI (Message-Passing Interface) [7]. Implementations of MPI are now widely available, including portable and freely available implementations [2, 3, 8] and specialized versions from vendors. General information on MPI is available at [1]. For the purposes of this paper, it will be useful to refer to the result of the initial MPI standardization effort as "MPI-1."

MPI-1 defined an interface for a specific *message-passing model* of parallel computation, in which a fixed number of processes with disjoint address spaces communicate through a cooperative mechanism (when two processes communicate, one sends and the other receives). MPI provides many types of point-to-point communication, to incorporate requirements for robustness, expressivity, and performance. Messages are strictly typed and scoped, allowing for communication in a heterogeneous environment. MPI also contains an extensive set of collective operations, process topology functions, and a profiling interface.

The most distinctive feature of the current MPI-2 proposals described in this paper is that they go beyond the strict message-passing model defined above. In MPI-2, processes may create other processes, so that the number of processes in an MPI computation can change dynamically (Section 2). Processes can interact directly with the memory of other processes (Section 3). Extensions, semantic modifications, and subset definitions in support of real-time and embedded systems (Section 4) also represent changes to the computational model.

Other topics being discussed in MPI-2 include extending MPI-1's collective operations to intercommunicators and nonblocking operations (Section 5), bindings for C++ and Fortran 90 (Section 6), and interface definitions for some of MPI's opaque objects so that they can be used more effectively in support of profiling and other libraries (Section 7). Finally, a number of issues, such as interlanguage communication, a portable startup mechanism, and minor repairs to the MPI-1 specification (Section 8), are under consideration in MPI-2.

In the rest of this paper, we present an overview of each of these areas. We assume familiarity with the current MPI Standard. In the Conclusion we describe the current status of these proposals and prospects for their early appearance in implementations.

2 Dynamic Process Management

MPI-1 describes how a group of processes can communicate with one another. It does not specify how those processes are created, nor does it allow processes to enter or leave a parallel application after the application has started. This static process model enables the specification of deterministic semantics and facilitates efficient implementations of MPI.

Nevertheless, a number of important applications cannot use MPI-1 because of the constraints imposed by its static process model. These include manager-worker applications, where the number and type of workers are not known until the manager has started, task farms, applications that can adapt to changing resources, applications with varying resource requirements, and client/server applications. Much of the impetus for relaxing the static process model comes from the PVM community, which is familiar with PVM's relatively rich support for dynamism.

2.1 The Interface

A fundamental concept in MPI-1 is `MPI_COMM_WORLD`, which defines the communication space containing all processes in an MPI application. With MPI-2's ability to add more processes to an application, the definition is modified to be the communication space containing all processes started together. Groups of newly started processes each have their own unique `MPI_COMM_WORLD`, but they also have an intercommunicator that allows them to merge with their parent group, forming a single bigger communicator. MPI-2 also provides an attribute,

MPI_UNIVERSE_SIZE, that suggests how many new processes might usefully be spawned in the environment.

A powerful new functionality being added to MPI-2 is the ability to establish contact between two groups of processes that initially do not share a communicator and may have been started independently. This functionality would be useful, for example, in enabling a visualization tool to start up and attach to a running simulation, or in enabling two parts of a large application, started separately at two different sites to communicate with each other. The collective functions MPI_CONNECT and MPI_IACCEPT create an intercommunicator that allows the two groups to communicate.

3 Remote Memory Access

The message-passing communication paradigm requires explicit involvement of two processes (sender and receiver), in order to transfer data from the memory of one to the memory of another. Remote Memory Access (RMA) extends the communication mechanisms of MPI by allowing the transfer to occur with the explicit involvement of only one of the two processes.

3.1 Motivation

Remote memory access facilitates the coding of some applications with irregular communication patterns. One situation occurs when a distributed-memory application needs some randomly accessed read-only shared memory (for large shared tables). Some of the processes can be used as "memory servers", while the other processes access the data by using get calls. Another situation occurs with a distributed-memory code where the data distribution is fixed or slowly changing, but where the pattern of use changes dynamically. Each process can compute what data it needs from remote processes and generate the required receives. To generate the matching sends, one needs to compute the inverse of the receive mapping, a time-consuming process that requires all processes to coordinate the data exchange. The use of get calls avoids the need for sends. A generic example is the execution of an assignment of the form `A = B(map)`, where `map` is a permutation vector, and `A`, `B`, and `map` are distributed in the same manner.

RMA can be supported on distributed memory systems by an "RMA agent" at the target node that accepts RMA requests and performs the required read or write accesses in the memory of the target process. A portable implementation might use an asynchronous receive handler to implement this RMA agent. Systems with dedicated put/get hardware (for example, the Cray T3D) could take advantage of that hardware, at least for simple transfers. Systems with communication coprocessors can take advantage of that coprocessor in order to run the RMA agent without interfering with the application processor at the target node. On shared-memory systems, if the caller can directly access the memory of the target process, RMA can be implemented without an RMA agent: the caller process can directly copy data to or from the memory of the target process.

3.2 Interface Summary

The current MPI-2 draft proposes the following RMA operations:

Put: transfer data from caller memory to target memory
Get: transfer data from target memory to caller memory
Accumulate: update variables in target memory by values from the caller memory. The update operation is an associative operation such as addition or minimum.
Read-Modify-Write: update variables in target memory by values from the caller memory, and return the initial value of the target memory variables. With a suitable choice of the update operation, one obtains synchronization operations such as test-and-set, fetch-and-add, or compare-and-swap.

In addition, a generic asynchronous handler mechanism is provided. This mechanism can be used for a software implementation of remote memory access, as well as for implementing many other communication paradigms. However, the very generality of this mechanism prevents many implementation optimizations that are possible for the more specific RMA operations.

4 Real-Time Extensions to MPI

MPI has helped to promote performance-portable programming of traditional high-performance computing and cluster systems. It has also proven desirable to leverage the success of MPI on parallel applications in the real-time community.

Taking advantage of this opportunity, a number of new organizations and the existing MPI Forum participants initiated an effort to explore what "real-time MPI" might look like. It is not expected that real-time MPI will be a required part of the MPI-2 Standard or that all HPC and cluster MPI implementations will support the real-time profiles.

Time-Based Profile For the time-based profile, it has been tacitly accepted that an outside calendar must be provided, in addition to the MPI services, in order to schedule the computations associated with this profile of MPI/RT. The calendar will specify when to start MPI communication. The anticipated strategy is to extend the MPI interface by using persistent communications that support this timed startup of communication. Timeout-based communication also will be supported in this way.

Priority-Based Profile Priority-based messaging and threading are commonly occurring strategies in real-time and non-real-time systems. Priority levels are supported by various operating systems and by certain message-passing networks, though not widely. Furthermore, some network systems support virtual channels, which themselves may provide a mechanism of reservation, if not priority, for given "flows" of data.

5 Collective Communication Extensions

MPI-1 has a rich set of collective operations, but they are subject to a number of restrictions. MPI-2 is considering generalizing them is a number of directions.

Asynchronous Operations In the current draft, each collective operation specified by MPI-2 has an asynchronous analog. A wide variety of MPI-2 features use asynchronous collective operations on both intracommunicators and intercommunicators.

Intercommunicator Collective Operations The purpose of intercommunicator collective operations is to support broadcast, reductions, and other operations, extended to include the two-group model of parallel processing offered in MPI-1 by intercommunicators.

Original proposals for extending intercommunicators to support collective operations, in addition to their MPI-1 point-to-point facilities, were first based on [10], which included model implementations.

The additional functionality came in three forms: more collective constructors and manipulators, what is now called "half-duplex" intercommunicator operations that extend intracommmunicator collective operations, and virtual topology-oriented versions of both the constructors and the communication procedures.

6 Language Bindings

6.1 C++ Bindings

The design of MPI itself is very much object-based, and the C++ bindings are based on the underlying object-based design principles. The bindings define a small set of classes corresponding to the fundamental object types in MPI with the functionality of MPI provided as member functions of these objects. This interface is fairly lightweight and seeks to meet the requirements of a language binding while still using advanced features of the target language. For instance, MPI error codes are still returned by function calls, no new types of objects are introduced, and the type arguments to function calls must be explicitly provided. Thus, only minimal use of advanced features of C++ such as polymorphism would be available to MPI programmers. This is an approach similar to that taken in [6]. A full-fledged class library that uses such advanced features has been developed in conjunction with the bindings and can be found at [9].

6.2 Fortran 90 Interface

Fortran 90 adds a wide range of features to Fortran 77. These include the module facility, derived types, array syntax, dynamic memory allocation, "pointers", the ability to do strict type checking, and function overloading. At first glance,

it seems that MPI-2 should be able to make wide use of these new features. Unfortunately, most of them are too "high level" for MPI to use, and many in fact cause more problems than they solve. The MPI-2 approach to Fortran 90 bindings therefore focuses more on trying to avoid introducing new problems than on trying to solve old ones.

7 External Interfaces

MPI-1 has a number of features that allow users to layer various capabilities on top of MPI. For example, user-defined reduction operations allow the programmer to use MPI for all communication requirements but still perform specialized reduction operations.

Generalized Requests MPI-1 had nonblocking operations for basic point-to-point send and receive calls. MPI-2 is proposing nonblocking calls for all collective operations, many one-sided operations, and dynamic spawning. Although these significantly expand the areas covered by nonblocking operations, users still may want additional nonblocking operations. For example, in the current MPI-IO effort [4, 5], nonblocking read and write operations are proposed. It would be advantageous to offer a standard MPI mechanism to perform these additional nonblocking operations. This would allow the use of other MPI features such as MPI_WAIT, reducing the effort in creating such requests and allowing one to control both types of nonblocking operations together.

Access to Opaque Objects One area that has caused difficulties in writing portable tools is the information stored with opaque objects. MPI-1 was deliberately designed with opaque objects. These allow flexibility in implementations and allow for future enhancements without changing the user's view of objects already present in MPI. To allow users to gain access to needed information in opaque objects, MPI has a number of accessor functions. For example, `MPI_GET_COUNT` will return the number of entries received as stored in the opaque part of the status object. One drawback to this approach is that only information with explicit accessor functions can be obtained in an easy and portable way from an MPI implementation. In MPI-1, the MPI Forum included all the accessor functions that seemed to be needed by users. However, tool writers have noted that they need access to information not typically needed by users. For example, a profiling library often needs the length of a message begun by `MPI_START` for a persistent request. To enable these tools to be truly portable, MPI-2 includes a number of functions to expose information stored in opaque objects.

Finally, the external interface definition in MPI-2 allows a generalization of the MPI-1 caching mechanism to allow caching on additional handles. The same calls are used but in MPI-2 apply to `MPI_COMM`, `MPI_DATATYPE`, and `MPI_GROUP`.

8 Miscellaneous

A number of topics are being considered by the MPI-2 Forum that do not fall into the categories above.

In MPI-1, although both C and Fortran-77 bindings were defined, nothing was specified regarding the interoperability of these two languages. Interoperability comprises at least three subareas: initialization, passing of MPI opaque objects from one language to another, and sending a message from one language and having it received in the other.

Only one form of `MPI_INIT` need be called. After the call, the MPI library will be completely initialized for all supported languages.

In order to deal with the portability of MPI opaque objects, such as datatypes, communicators, and requests, conversion functions will be provided that convert the language-dependent "handles" to 32-bit integers and back again. These integers will be portable (among languages) versions of the objects they reference.

Sending a message from a Fortran program to a C program or vice versa will be explicitly allowed, as long as the signatures of the datatypes match. Here we are aided by the fact that the elementary datatypes defined in MPI-1 are distinct in the two languages, and no equivalence (such as one that might exist between the C datatype `int` and the Fortran datatype `INTEGER` on some machines) is assumed. Thus, in sending messages between programs written in different languages, one sends data of a given MPI datatype; no automatic conversion takes place.

9 Conclusion

We have described the current state (February, 1996) of MPI-2 discussions. The precise content of MPI-2 remains to be decided in the coming months. Although a few implementors are beginning to experiment with some of the notions described here, most are waiting to see what the final specification will look like. The MPI-2 features will be more difficult to implement than those of MPI-1. Nonetheless, enough discussion has taken place that it is possible to discern the likely scope of the functionality that MPI-2 will add to MPI-1. In this paper we have described that functionality.

References

1. World Wide Web MPI home page.
   `http://www.mcs.anl.gov/mpi/standard.html`.
2. R. Alasdair, A. Bruce, James G. Mills, and A. Gordon Smith. CHIMP/MPI user guide. Technical Report EPCC-KTP-CHIMP-V2-USER 1.2, Edinburgh Parallel Computing Centre, June 1994.
3. Greg Burns, Raja Daoud, and James Vaigl. LAM: An open cluster environment for MPI. In John W. Ross, editor, *Proceedings of Supercomputing Symposium '94*, pages 379–386. University of Toronto, 1994.

4. Peter Corbett, Dror Feitelson, Yarsun Hsu, Jean-Pierre Prost, Marc Snir, Sam Fineberg, Bill Nitzberg, Bernard Traversat, and Parkson Wong. MPI-IO: A parallel file I/O interface for MPI, version 0.3. Technical Report NAS-95-002, NAS, January 1995.
5. Peter Corbett, Yarsun Hsu, Jean-Pierre Prost, Marc Snir, Sam Fineberg, Bill Nitzberg, Bernard Traversat, Parkson Wong, and Dror Feitelson. MPI-IO: A parallel file I/O interface for MPI, version 0.4. `http://lovelace.nas.nasa.gov/MPI-IO`, December 1995.
6. Nathan E. Doss, Purushotam V. Bangalore, and Anthony Skjellum. MPI++ : Issues and Features. In *Proceedings of OONSKI '94*, January 1994.
7. The MPI Forum. The MPI message-passing interface standard. `http://www.mcs.anl.gov/mpi/standard.html`, May 1995.
8. William Gropp and Ewing Lusk. *User's Guide for* `mpich`, *a Portable Implementation of MPI.* Argonne National Laboratory, 1994.
9. Andrew Lumsdaine, Brian M. McCandless, and Jeffrey M. Squyres. Object-oriented MPI, 1996. `http://www.cse.nd.edu/lsc/research/oompi/`.
10. Anthony Skjellum, Nathan E. Doss, and Kishore Viswanathan. Intercommunicator extensions to MPI in the MPIX (MPI eXtension) Library. Technical report, Mississippi State University — Dept. of Computer Science, April 1994. Draft version.