EARLY APPLICATIONS IN THE MESSAGE-PASSING INTERFACE (MPI)

Anthony Skjellum

DEPARTMENT OF COMPUTER SCIENCE
NSF ENGINEERING RESEARCH
CENTER FOR COMPUTATIONAL
FIELD SIMULATION
MISSISSIPPI STATE UNIVERSITY
MISSISSIPPI STATE, MISSISSIPPI 39762

Ewing Lusk
William Gropp

ARGONNE NATIONAL LABORATORY
MATHEMATICS AND COMPUTER
SCIENCE DIVISION
ARGONNE, ILLINOIS 60439

Summary

We describe a number of early efforts to make use of the Message-Passing Interface (MPI) standard in applications, based on an informal survey conducted in May–June, 1994. Rather than a definitive statement of all MPI developmental work, this paper addresses the initial successes, progress, and impressions that application developers have had with MPI, according to the responses received. We summarize the important aspects of each survey response, and draw conclusions about the spread of MPI into applications.

An understanding of message passing and access to the MPI standard are prerequisites for appreciating this paper. Some background material is provided to ease this requirement.

The International Journal of Supercomputer Applications, Volume 9, No. 2, Summer 1995, pp. 79–94
© 1995 Massachusetts Institute of Technology.

1 Introduction

In this paper we describe a number of early efforts to make use of the Message-Passing Interface (MPI) standard in real applications (MPI Forum 1994a, b). An informal survey of efforts is reported here, together with our commentary. We summarize the responses, highlighting important facets of each contributed application survey, and give an overall impression of the state of MPI applications.

1.1 A BRIEF HISTORY OF MPI

MPI was developed as part of a multinational effort (supported partially by the U.S. National Science Foundation, Advanced Research Projects Agency, and the European Strategic Programme for Research and Development in Information Technology) to replace the many, marginally compatible vendor and portability systems currently in use with a coherent system that exploits the experience of these previous message-passing systems while providing greater functionality, unified design, and an emphasis on user access to high-performance protocols and hardware. Participants included a core of 40 or so forum members, including developers of all the major message-passing systems then in existence (e.g., p4, PVM, NX), as well as computer vendors and application developers. MPI encompasses and extrapolates the best ideas from existing practice and adds features designed to secure high performance for applications on the hardware that exists today and for hardware expected in the next few years.

The initial round of that MPI standardization effort began in April 1992 and culminated at Supercomputing '93 (November, 1993), where the standard was presented to the public.[1] By the time of its public introduction, several implementations were already under way, notably the Argonne/Mississippi State "model implementation" (also known as MPICH) (Doss et al., 1993), which was also demonstrated at Supercomputing '93; other portable implementations have followed (e.g., LAM (Burns, Daoud, and Vaigl, 1994), and CHIMP

1. However, the standard document was finalized only in May, 1994, with on-going errata, so the existence of several applications at the writing of this paper in June, 1994 is quite significant.

MPI (Edinburgh Parallel Computer Centre)). During the first half of 1994, IBM launched a commercial MPI effort (Franke et al., 1993); IBM demonstrated its experimental MPI-F at Supercomputing '93 as well. Cray Research, through an alliance with the Edinburgh Parallel Computing Centre, is working to support MPI on the CRAY T3D. *n*CUBE has made MPI the native message-passing system for its next generation systems. Intel, Meiko, and others also have implementation efforts or plans; the MPICH and other portable implementations run on virtually all the machines for which commercial versions remain to be created.

1.2 SOME FEATURES OF MPI RELEVANT TO APPLICATIONS

In order to make our discussion more accessible, it is helpful to describe certain features of MPI. The first of these is the dual concept of process groups and communicators. Here we draw on the results of MPI Forum 1994a,b.

Groups and Communicators. MPI designates communicating processes with communicators (intra-communicators and inter-communicators). An intra-communicator consists of a group of processes, described first, and "context" described thereafter. An MPI program currently gets the communicator MPI_COMM_WORLD when it starts executing; this communicator defines the initial group and communication space for processes to use. Other communication relationships can then be created during the course of program execution.

A group is an *ordered* collection of processes. Processes in a group, therefore, have a unique integer rank (starting from zero). Once created, a group never changes size (hence, MPI groups are sometimes called "static groups"). A process group provides the rank property for send-and-receive communication and indicates which processes participate for collective communication (synchronization, global sum, etc.). MPI's concept of designating groups as boundaries for communication is extremely important to its ability to scope messages. Scoping of messages allows any parallel code to insulate itself from user code or other libraries in the same application.

Intra-communicators include the notion of message context. A context keeps messages sent by a process from arriving in the wrong scope of a receiving process. Think of planes of message passing, with processes belonging to as many hypothetical planes as makes sense for application requirements, but any message is restricted to a single plane. This naming convention leads to simple scoping rules for parallel libraries with simple-to-use techniques for assuring that a library has safe communication space (lexical scoping of messages). Finally, communicators guarantee that pending sends and receives cannot foul up a collective communication operation. This further simplifies parallel programming. Intra-communicators are the dominant programming idiom at present, so the word "communicator" is often used loosely to refer to them.

An inter-communicator involves a "local" and a "remote" group, and appropriate context information, and is consequently a simple generalization of the intra-communicator. The intra-communicator allows a local set of processes to describe point-to-point communication to a remote group, and vice versa. MPI emphasizes point-to-point communication for inter-communicators, though extensions are possible (Skjellum, Doss, and Viswanathan, 1994).

Virtual Topologies. Virtual topologies allow the application programmer to attach numerical names other than group ranks to processes, so that the domain decomposition or other application-relevant mapping of data to processes is intuitive and convenient. MPI provides such mappings for the convenience of the application programmer; both Cartesian grids and general graph mappings are supported. As examples, Cartesian grids of one-, two-, and three-dimensions are quite useful for linear algebra codes and solution of PDEs. Graph topologies might be useful for the solution of problems with sparse connectivity or for combinatorial problems (see Gropp, Lusk, and Skjellum, 1994, for more details).

Datatypes. MPI provides basic datatypes as part of communication calls. In the simplest case, the first-class types of each language are available as datatypes to perform sends, and end receives, as well as collective communication calls. MPI programmers send arrays of types and do not normally send arrays of bytes, unless they choose to do so. These datatypes encapsulate any heterogeneous conversion needed. Hence, an MPI programmer does not worry about XDR conversion, as this is part of the user program (in fact, something faster than conventional XDR is likely in efficient implementations).

In addition to predefined datatypes, MPI supports aggregate or derived datatypes (ideally defined once and used repeatedly). These allow arbitrary gather-scatter patterns to be created. In this mode, the application programmer does not explicitly "buffer up" data into a contiguous area of memory prior to calling MPI send, nor does the programmer have to scatter a contiguous buffer on the receive end, unless they prefer to work in this pattern. Such a strategy can reduce the number of copies needed to effect data transfer. As such, it both simplifies and possibly speeds up message passing.

"Communicators guarantee that pending sends and receives cannot foul up a collective communication operation. This further simplifies parallel programming. Intra-communicators are the dominant programming idiom at present."

1.3 APPLICATIONS REPRESENTED

Application programmers were informed of the opportunity to contribute to this paper through an informal survey mechanism published on popular mailing lists, and in USENet groups. The responses described here represent all but one of the respondents during the period allotted for the survey (one response was too preliminary to contain any reasonable information about the work undertaken). Responses came in the following categories:

- Applications:
  - many-body methods for light nuclei
  - solution of unsteady incompressible viscous flows
  - NAMMU—groundwater modeling for continuous media
  - PARAMICS-MP—microscopic traffic simulation
  - volume visualization
  - reservoir simulation (not included)
- Application-Enabling Libraries:
  - fast particle summation library
  - PRISM Eigensolver library
  - CVL—C Vector library
  - ParX (parallel X environment)

"We asked respondents to provide anecdotal information describing the process of converting from other message-passing systems, including their level of effort, motivation for conversion, and degree of satisfaction."

1.4 ORGANIZATION

This paper is organized as follows. First, we outline the survey itself. Then, each of the surveyed applications is described; for clarity, we have not followed the format of the survey in reporting the application responses. Next is an interpretation of the responses to the survey, followed by a summary, conclusions, and discussion of future directions.

2. The Survey

To collect data from the widest possible source of application programmers, consistent with a short deadline, we posted requests for application work in major USENet newsgroups (comp.parallel, comp.parallel.pvm), as well as to the MPI reflectors maintained by the University of Tennessee (mpi-comm@cs.utk.edu) and Argonne National Laboratory (mpi-users@mcs.anl.gov). Responses were solicited for applications that were under way, completed, or even just getting started.

The following information was requested for contributions to the survey: a description of the MPI-based application, areas of interest of researchers who were actively involved, and the experiences thus far. In addition, we requested a description of the application area, the goals of the application project (high speed, cheap cycles, new physics, and so on), proper citations to the project, and the techniques employed in the project.

We asked respondents to provide anecdotal information describing the process of converting from other message-passing systems (if applicable), including their level of effort, motivation for conversion, and their degree of satisfaction. We asked them to determine which features of MPI have proven most helpful in applications development and/or conversion, such as

- intra-communicators (single-process group communication)
- inter-communicators (two-process group communication)
- virtual topologies (naming conventions for processes)
- pack/unpack capabilities (PVM compatibility)
- datatypes (gather-scatter specifications)

We also asked respondents to state those parts of MPI that were most understandable and those that were the most confusing. We asked them to show early indications of performance results and specify which MPI implementation or implementations they have used in conjunction with the applications. Finally, respondents were invited to provide miscellaneous feedback from application programmers and users, if relevant. For instance, we asked them to discuss whether they felt there was a need for revised and/or additional features in MPI.

3 Applications

In this section, we describe five responses to the survey, each a pure application project.

3.1 MANY-BODY METHODS FOR LIGHT NUCLEI

Researchers (Steven Pieper and Robert Wiringa) in Argonne's Physics Division, in collaboration with Vijay Pandharipande at the University of Illinois at Urbana-Champaign, are computing the properties of light (up to 40 neutrons and protons) nuclei using realistic two- and three-nucleon interactions. This research involves developing many-body methods for reliably computing the properties of a nucleus for complicated forces that are strongly dependent on the spins and charge states of the nucleons.

Unlike the Coulomb force used in atomic or condensed-matter calculations, there is no useful fundamental theory that tells us what this force might be. The two-body force can be partially constrained by fitting nucleon-nucleon scattering data, but many-body calculations are required to test other properties of this force as well as the three-body interaction. Thus the researchers are refining their knowledge of the forces and at the same time using this knowledge to make predictions about nuclei.

3.1.1 GOAL OF PROJECT

The goal of the application project is to reveal new physics: that is, to compute binding energies more accurately than before. New calculations being done on the IBM SP1 parallel computer at Argonne are using a new nuclear interaction and are obtaining much better results for the binding energy and density profile of oxygen than had previously been obtained in experiments. In particular, physicists are now able to find the values of the parameters of the minimal-energy solution for 16 nucleons much more precisely than ever before. Before this computation, theory did not clearly predict that the nucleus of an oxygen atom would remain stable; the previous computational results would have allowed it to disintegrate into four helium atoms. Work on the calcium atom (40 nucleons), which could not be contemplated before, is now also in progress.

3.1.2 DETAILS

The program was originally written for portable parallel execution using p4; it took one afternoon to convert it to MPI. The authors' motivation was to learn about MPI and prepare the code for continued portability on high-performance machines as further MPI implementations become available. The parallel algorithm used was straightforward as no advanced features of MPI were needed. Understandably, the most helpful aspect of MPI was that it is similar to existing message-passing systems, rendering porting straightforward as well.

This application used the portable model MPI implementation developed at Argonne National Laboratory and Mississippi State University. One feature that the p4 implementation provided that was unavailable in MPI is a form of broadcast that is received by an ordinary receive instead of a corresponding broadcast operation; this was missed by the developers. Table 1 contains information about the size and performance of this application on the 128-node SP-1 at Argonne.

3.2 SOLUTION OF UNSTEADY INCOMPRESSIBLE VISCOUS FLOWS

Computation of flow regimes over complex configurations entails the numerical solution of a system of coupled nonlinear partial differential equations, the Navier-Stokes equations. Researchers in the Computational Fluid Dynamics Laboratory at the Mississippi State University-NSF Engineering Research Center for Computational Field Simulation have developed an implicit, finite-volume code (known as UNCLE) for solv-

Table 1
Summary of the Argonne Nuclear Structure Code

Lines of Fortran code and comments	23,000
Lines in time-critical subroutines	1000
Total program single-processor speed (megaflops)	48
Total 128-node speed (gigaflops)	5.9
Speed in the critical subroutines (megaflops)	36–56

ing the unsteady three-dimensional incompressible Euler and Navier-Stokes equations using dynamic multiblock grids (Taylor et al., 1993; Taylor and Whitfield, 1991; Whitfield and Taylor, 1991). The flow solver can be used in a variety of applications ranging from maneuvering underwater vehicles to the design of centrifugal compressors. Ramesh Pankajakshan and W. Roger Briley have undertaken the scalable parallelization research and development of this solver.

Long runtimes and large memory requirements restrict the size and complexity of the problems that can be handled using the sequential version of the UNCLE code. Therefore, a scalable portable parallel version is being developed that can take advantage of existing as well as emerging parallel platforms. The goal is to use parallel computing to enable solution of large-scale field simulation problems on both parallel supercomputers and idle and/or dedicated workstation clusters. The message-passing interface required for the parallel implementation has to support collective operations within user-defined groups as well as provide safe communication contexts for overlapping sets of collective operations. Motivations for the use of MPI include these features, as well as portability, ease of software development, and high-performance expectations, while the broad base of participants in the MPI effort guarantees continued access and evolution as hardware progresses.

3.2.1 DETAILS

The parallel formulation (Pankajakshan and Briley, 1994) employs spatial decomposition of the overall grid into sub-blocks assigned to separate processes, and exploits coarse-grained parallelism and message passing within sub-iterations of an implicit time-dependent solution algorithm. The solution at points shared by neighboring processes is updated between each sub-iteration by means of a message exchange.

Key areas of the parallel implementation include:

- initialization of the flow field
- duplication of stored data for points near block interfaces
- exchange of data during sub-iterations for points having duplicated storage

- treatment of line searches and integrals along coordinates emanating from solid boundaries (the latter arise from the particular algebraic turbulence model used)

In the parallel implementation of this code (ibid.), the domain is partitioned into several nearly equally sized sub-domains, each of which is assigned to a node of the multicomputer. The local data dependencies at the boundary of each block are taken into account by a two-cell-deep layer of buffer cells whose values are updated from the appropriate neighboring block. Data duplication and updating at the block boundaries are implemented using the MPI_Sendrecv routine. The connectivity of the processes gives rise to a Cartesian virtual topology with empty nodes. Each process is tagged using an ordered triplet representing its coordinate on the virtual Cartesian grid (topology). These coordinates are then used to define communicators for processes that are oriented along any of the three axes of the grid. This involves repeated use of the MPI_Comm_split routine. The line searches and integrals of the turbulence model are implemented using the MPI_Bcast routine and user-defined MPI_Allreduce operations over subsets of processes.

Work has progressed to the point of a working (but as yet preliminary) MPI application. Constructing this MPI code from an existing serial Fortran–77 code took about three weeks by one of the researchers (Pankajakshan) with a working knowledge of MPI and C programming and who was already familiar with the existing serial code. From their experience, they have found that MPI appears well suited for this application. It was especially useful in performing collective operations on subsets of processes. This is important in line searches and integrals in the algebraic turbulence model. Communicators and datatypes are features of MPI that have been or will be helpful in developing this application. However, datatypes were seen as the most confusing aspect of MPI.

The parallel code has used both the IBM's MPI-F and Argonne/Mississippi State implementations of MPI. The code has run on the IBM-SP1 at the High Performance Computing Research Facility, Mathematics and Computer Science Division, Argonne National Laboratory and on an MSU Sun workstation/ethernet cluster.

3.3 NAMMU—GROUNDWATER MODELING

NAMMU is a package for predicting groundwater flow, heat, and mass transport through continuous porous media. The application is based on an efficient implementation of the finite-element method, which in turn makes heavy use of the Harwell Frontal Direct matrix solver routine MA42. The work described here is a parallelization of the solver through domain decomposition of the mesh. NAMMU was developed by AEA Technology, while modification of the MA42 package was done by Harwell, and the message passing (in MPI) was done by the Edinburgh Parallel Computing Centre. NAMMU was written by Hon W. Yau (Edinburgh Parallel Computing Centre), K. Andrew Cliffe (AEA Technology, Harwell), Jennifer A. Scott (Atlas Centre, Harwell), and David Brear (AEA Technology, Harwell). Hon Yau is the designer (and one of the implementors) of the parallelized version of NAMMU.

The goal of this work is to produce a portable, parallelized version of NAMMU that best exploits parallel architectures. Together with two other AEA Technology safety assessment codes, this will form a suite of parallelized groundwater simulation codes.

3.3.1 DETAILS

The conversion effort was found to be considerably eased by the availability of a high-quality in-house implementation of the MPI specifications. Moreover, a 'toy' MPI implementation of the solver using MA42 had already been successfully demonstrated prior to work on the NAMMU package. It took approximately three months to go from the design of the NAMMU implementation to successful execution of the available test cases, including roughly 20 person-days for the conversion of NAMMU to the use of domain decomposition, and one person-month to write and debug the necessary MPI message-passing constructs.

The purpose in converting to MPI was to achieve a portable implementation that would demonstrate good scalability within the limitations of the parallel algorithm. A high degree of satisfaction with the current MPI specifications was reported, particularly as regards the abundance of features, and on the whole it was felt that MPI was a thoroughly well-thought-out product. Note that only a duplicate of the global group of processes (represented in MPI_COMM_WORLD) was used in the application, since the intention was to use one or more of the collective communications functions. Derived datatypes were heavily used as an efficient alternative to multiple messages and packing/unpacking; this feature had much to be commended.

The most obvious source of problems has to do with the Specifications Document (MPI Forum 1994a). As the only then available introduction to MPI it is seen by the developers as daunting—particularly to those with no prior message-passing experience. It is, however, also seen as reasonably complete, though in need of more Fortran examples and a glossary of terms.

An in-house EPCC implementation of MPI based on CHIMP was used. This version of MPI runs on a variety of machines, including Sun and Silicon Graphics workstations, as well as the Meiko CS-1/i860.

3.4 PARAMICS-MP—MICROSCOPIC TRAFFIC SIMULATION

PARAMICS-MP builds on earlier work in the PARAMICS project in which a PARAllel MICroscopic traffic Simulator was developed. In its current state, the PARAMICS system can simulate over 250,000 vehicles at speeds faster than real time, on a road network based on real data for the Scottish Trunk Road Network. According to the researchers, this is the largest microscopic traffic simulation ever implemented. PARAMICS-MP is being developed by Gordon Cameron, Gordon Duncan, and David McArthur of the Edinburgh Parallel Computing Centre.

The primary goal for PARAMICS-MP is to port the existing PARAMICS simulator, originally developed as a data-parallel implementation for the Connection Machine CM-200, to a message-passing version running on the recently purchased 256-processor CRAY T3D located in Edinburgh. The MP version will use the MPI library currently under development at EPCC.

It is expected that the T3D's superior computational power will enable integration of a significantly more detailed vehicle-dynamics model to the simulator than is currently possible. In addition, more advanced simulation features will be added, such as the effects of dynamic routing as proposed by reactive road traffic information systems.

Currently a prototype version of the MP simulator is running on a distributed system using EPCC's own message-passing system CHIMP. The application is to be connected to the MPI implementation being developed locally for the T3D as soon as it becomes available. This move is part of an EPCC organization-wide policy of migration to the MPI standard in all forthcoming applications.

3.5 VOLUME VISUALIZATION

Volume visualization techniques were created to allow researchers to examine the data and spatial relationships contained in volume data sets, but, in general, the algorithms are relatively slow on even moderately sized data sets. The problem is compounded with increasing data size, and additional problems with memory size may be encountered in generating a visualization for large data sets on a conventional machine. It is thought that some of these problems can be addressed by distributed memory parallel computers, by allowing the parallel machine to perform the calculation of the scene and employing conventional machines to display that scene. Parallel computers have added potential over conventional machines for scene rendering in that a large data volume can be distributed over the processors in the machine and operated on concurrently, reducing at once the computational time and memory requirement per processor (scalability) (Lorensen and Cline, 1987; Westover, 1990; West, Stephens, and Turcotte, 1994b; West, Stephens, and Turcotte, 1994a).

John E. West, Michael Stephens, and Louis Turcotte of the Waterways Experiment Station (WES) are investigating the application of MPI to the development of algorithms for performing volume visualization of large data sets on parallel machines. The goal is to create preliminary versions of volume visualization algorithms for parallel machines both to investigate the use of a specific machine for this type of application, the nCUBE/2 at WES, and to gain insight into the issues surrounding parallel visualization in general. The algorithms under examination represent the two general classes of algorithms available for extracting information out of 3-D data sets: marching cubes, which is prototypical of surface-extraction algorithms, and splatting, a volume-rendering technique.

It is expected that after the preliminary study is completed, some or all of the algorithms will be ported to other parallel machines and evaluated for hardware-specific performance features. Using MPI it is expected that the cost of this software port will be reduced dramatically and code development simplified.

3.5.1 DETAILS

Conversion issues do not apply to this project, since the code is being created from scratch using MPI. According to the researchers, this represents a distinct advantage, in that not only will this reduce the initial development time, but also reduce the time it will take for subsequent ports of the code to different platforms.

Thus far, features of MPI that have proven most helpful are communicators and topologies. Having developed codes for varying platforms in the past using native message-passing constructs (NX, Vertex, etc.), these researchers state the communicators and utility functions available for their manipulation greatly simplify the development of a communication protocol within the application. The topology functions have also been useful in performing data distribution, and the authors have suggested that a process topology matching the original data topology could be created in only a few simple steps. This further simplifies the data distribution and communication problems typically encountered in other message-passing protocols. It is expected that datatypes, and, in particular, strided datatypes, will also prove especially useful in the future.

"Thus far, the features of MPI that have proven most helpful are communicators and topologies. The topology functions have been useful in performing data distribution. It is expected that data types will also prove especially useful in the future."

4 Application-Enabling Libraries

In this section, we describe application-related responses to the survey. Each of these responses represents a project that is either part of an application, or enables a class of applications.

"*Of all the features the authors were interested in, the multiple completion functions proved to be much less flexible than they desired. They would have preferred functions that act upon dynamic sets or collections of communication objects which can be updated efficiently.*"

4.1 A FAST PARTICLE SUMMATION LIBRARY

A general-purpose Fast Particle Summation library that is in turn used in several applications,[2] is under development by John K. Salmon of Caltech, Michael S. Warren of Los Alamos and David J. Edelsohn of Syracuse University. The code itself and its original application in gravitational N-body simulations has been described by Salmon, Winckelmans, and Warren (1994), Warren and Salmon (1993), and Warren and Salmon (1994).

4.1.1 DETAILS

The application is currently written to the researchers' own set of message-passing primitives (e.g., asynchronous send, asynchronous receive, exchange, etc.). These primitives are, by design, extremely simple. They are implemented using the message-passing programming interface of each system. Because the primitives previously had been ported to the IBM MPL (EUI), converting to MPI was greatly simplified in view of the similarities between the two message-passing definitions. The most difficult element was in adapting to system peculiarities for querying about the status of outstanding operations and statistics about received messages. The researchers ported to MPI because it expanded their target environments and improved flexibility and maintenance for future systems on which this application will run, as well as being simple to accomplish. At the moment the authors only use the lowest level of MPI functionality. Using inter-communicators for interactive, real-time control and display of simulation results is their long-term goal.

Of all the features the authors were interested in, the multiple completion functions proved to be much less flexible than they desired. They would have preferred functions that act upon dynamic sets or collec-

2. A description of an application solving potential-flow problems and another involving generation of Gaussian random fields is in preparation. All the relevant papers, including some with an emphasis on astrophysics rather than computation are available by anonymous ftp in ftp://ftp.ccsf.caltech.edu/nbody, or on Mosaic (HTML) as http://www.ccsf.caltech.edu/johns/papers.html. Papers describing the evolution and interaction of disk galaxies by David Edelsohn (NPAC, Syracuse University) and Bruce Elmegreen (IBM Research) are in preparation.

tions of communication objects which can be updated efficiently and then used by a generalized POSIX select() function. This would have made possible the rapid discovery that one of many outstanding communication operations (there may be hundreds of such outstanding operations in their implementation) had completed. Their conclusion is that the current MPI array semantics does not allow for efficient implementation of the functionality they desire for their applications and is of limited usefulness for applications with a relatively short and mostly static list of communication handles.[3]

The IBM MPI-F prototype implementation running on an IBM SP1 with the High Performance Switch achieves a latency of 30 microseconds and a bandwidth of 8.7 MBytes per second. This implementation is an early research prototype developed within IBM Research that conforms to and implements the entire, final MPI document (MPI Forum 1994a). They observed no significant change in performance when they compared the application running with MPI-F and MPL/p (EUI-H) on the IBM SP1 (Franke, Hochschild, Pattnaik, and Snir, 1993).

4.2 CVL—C VECTOR LIBRARY

CVL (C Vector Library) is a library of low-level vector routines callable from C. It is used as a basis for implementing NESL, a portable nested data-parallel language (Blelloch et al., 1994) and also by the Proteus project at the University of North Carolina to implement the parallel prototyping language Proteus. The library includes a wide variety of vector operations such as elementwise function applications, scans, reductions, and permutations. Most CVL routines are defined for both segmented and unsegmented vectors, since segmentation is critical for implementing nested data parallelism. Previously, CVL has been ported to each new parallel machine architecture (CRAY, CM-2, CM-5, MasPar, etc.) using the manufacturer's own low-level communication routines. The main goal of this project is the development of a portable, interactive environment for programming a wide range of supercomputers, using a very high-level data-parallel language that supports nested parallelism. We are especially interested in the implementation of algorithms with irregular or dynamic structures.

The MPI port of CVL took approximately two person-months. It benefited from the use of algorithms for a distributed memory architecture initially developed for the CM-5 port, although these had to be adapted to use coarse-grain rather than fine-grain communication. The overwhelming motivation for writing an MPI port of CVL was the developer's wish to avoid having to perform any additional ports in the foreseeable future. Porting to any other machines in the future would only mean tuning the MPI port to best fit a particular manufacturer's implementation of MPI. The developer has been satisfied with MPI's portability and ease of use; this was declared to be the easiest of all the CVL ports to MPP architectures.

The developer notes that the most helpful features of MPI for the CVL implementations are as follows:

- built-in support for scans and reductions, although this is not as complete as it could have been
- non-blocking sends and receives to overlap communication and computation
- choice of send semantics with respect to blocking and buffering, making it easy to get a quick-and-dirty version running that can then be refined to increase performance.

Complete performance results are given by Hardwick (1994). MPI CVL comes close to the performance of a machine-specific CVL implementation on the CM-5, the only platform on which the developers can currently perform a comparison. Furthermore, the developer noted that better support in MPI for fine-grained communication (e.g., put/get, active messages, or dynamic all-to-all communication) would simplify the

3. In our experience, many applications do not have more than a handful of outstanding communication operations pending, unlike the algorithms considered by these researchers. The alternative functionality proposed by these researchers near the end of the MPI standardization process was consequently rejected as an alternative. Had it been proposed as a supplement to the easy-to-use arrays of status objects, rather than a replacement, its inclusion would have been much more likely. Undoubtedly, we will revisit this issue for MPI2.

"The developers have used communicators to allow our library to work in a safe communication space. They also use topologies to define a logical 2D grid/torus that they work on as well as row and column subgroups for collective communications."

code for MPI CVL and could also improve its performance. It was also noted that it would be helpful if MPI could support exclusive scans as well as the inclusive variety: it is trivial to convert an exclusive scan into an inclusive scan, but the reverse involves extra communication for operations with no inverse (e.g., max, min). Segmented scans and reductions, as supplied by High Performance Fortran, would also be an advantage, although these can be implemented together using communicators or user-defined scan operators in MPI.

4.3 PRISM—SCALABLE EIGENSOLVERS

The PRISM (Bischof et al., 1994) project is investigating scalable eigensolvers for parallel distributed-memory computers. The goal is to produce a public domain library that runs on a variety of platforms. To achieve this goal the project has looked at various issues including linear algebra kernels such as matrix multiplication and matrix factorization, data layout, and communication optimization. Christian Bischof, Steven Huss-Lederman, Xiaobai Sun, Anna Tsao, and Thomas Turnbull are working on PRISM. The PRISM project is aimed at producing a library to solve eigenproblems on a variety of distributed-memory computers. The library will be used by a variety of groups (e.g., computational chemists) to supply more computational power to the problem-solving process.

4.3.1 DETAILS

The developers found the effort needed to convert to MPI to be modest. A simple port was found to be reasonably easy. A slightly harder task has been taking advantage of features available in MPI which they could not previously access. The developers of the PRISM project wanted a package that runs efficiently on a large number of parallel systems rather than having to be tied to a specific vendor or national laboratory for message-passing software.

The developers have used communicators to allow our library to work in a safe communication space. They also use topologies to define a logical 2D grid/torus that they work on as well as row and column subgroups for collective communications. They have only made limited use of derived datatypes, because proper use of this feature requires modest rewriting of these sections of the code. However, they anticipate using derived datatypes to a greater extent in the future. They also plan on utilizing persistent communication handles since they have to perform operations such as shifting to the next processor repeatedly.

The developers have found the performance of MPI to be satisfactory and found that the performance of MPI improves with time. They are particularly interested in collective operations. Details can be found in the report by Bischof et al. (1995). The PRISM developers have used the MPI implementation from Argonne National Laboratory/Mississippi State (MPICH) on Sun clusters, Intel Delta and Paragon, TMC CM5, IBM SP1, and Meiko CS2. They have also used the MPI implementation from IBM (MPIF) on the SP1.

4.4 ParX

Steve Ball, of Australian National University, is the developer of ParX. ParX is a port of the X Window System to a distributed-memory multicomputer (in the first instance, the Fujitsu AP1000). Both the Xlib library and an X server are to be ported. This will allow an application running on the multicomputer to display graphics on a local framebuffer or a remote display using the same application programmer interface without having to off-load data to a host processor subsequently to generate X calls. There will be an implementation of the Xlib library allowing an application program to connect to an X server (local or remote) as well as an implementation of an X server for a distributed framebuffer. The server will itself be distributed across the multicomputer.

The developer was just starting a re-write of the Xlib implementation when X11R6 was released as an implementation of MPI for the AP1000. Thus no extra effort has been necessary to convert to MPI, apart from learning about MPI itself. In the not-too-distant future, the developer will be porting ParX's Xlib to the CM5 (CMMD), as CM5 users require MIMD X (as opposed to CMX11, which is SIMD). MPI allows code to be written portably to the CM5 as well as to other distributed systems. MPI has satisfied all of the developer's requirements, apart from the semantics of broadcasting which the developer believes could be overcome by a non-blocking, non-synchronizing broadcast. The developer noted that intra- and inter-communicators are vital for this project, as the project is actually a library that needs to insulate itself from the user's application. Pack/unpack may be useful later for optimizing memory access.

The point-to-point and collective operations are fairly straightforward (the author skipped over the datatype features). Communicators took a little while to work through, but they also turned out to be straightforward. The MPI release 0.1 for the Fujitsu AP1000 (port done by Australian National University) was apparently based in part on the Argonne/Mississippi State University portable implementation.

5 Interpretation of the Survey

We interpret responses here in three categories: conversion to MPI, target architectures, and needed improvements to MPI.

5.1 CONVERSION TO MPI

The applications we have surveyed reported a variety of effort levels in conversion. Conversion periods ranging from a few hours, to person-weeks, to person-months were reported. The applications used either C or Fortran–77, with no particular bias.

In addition to using simple features (send, receive), some of the applications exploited advanced MPI features, notably virtual topologies. Some applications used topologies, but only simple communicator strategies (copes of MPI_COMM_WORLD). Others exploited topologies as well as multiple communicators. In many cases datatypes were not used but those surveyed intended to use them in the future. We attribute this to the early stage of development of the implementations that were available when this report was written; in several cases datatypes were not fully implemented by the time the responses were received. However, respondents who tried datatypes found them almost universally confusing.

Although only limited performance results were described, it is clear that researchers are already getting good results on MPI-based applications. The light nuclei project of Pieper and Wiringa was the only one whose performance is quantified here, but it is evident that timings have been made in other projects too, for example, the Fast Summation Library, where researchers mention comparable performance to the fastest non-MPI message-passing system available on the IBM SP1. For other researchers, portability and development have preceded application tuning, so that detailed performance results will come only later. In any event optimization of MPI implementations will make performance results obtained in 1995 much higher than those that might have been quoted here.

Portability was a universal theme in all the responses received. Only positive comments were received concerning the process of converting to MPI, except for some researchers who had specific needs for collective operations that were not addressed in the initial version of MPI. Furthermore, the complexity of the datatype feature of MPI was shown repeatedly. The interface was seen to be confusing.

5.2 TARGET ARCHITECTURES

The variety of machines mentioned by the researchers was extremely interesting. Researchers target the IBM SP1, CM-5, and *n*CUBE/2, newer machines like the CRAY T3D, and less well known machines, like the Fujitsu AP-1000. All are currently able to run a portable version of MPI and some will have commercial versions of MPI in 1994, and many others in 1995. This indicates significant penetration of the MPI standard after only a few months. Furthermore, the variety of implementations of MPI in use (the Argonne/Mississippi State implementation, IBM implementation, and Edinburgh Parallel Computing Centre implementation) was significant. It is clear that at least two commercial MPI versions will emerge in 1995. Thus, application development is not at all premature in MPI.

It is interesting to note that several surveys mentioned using MPI on clusters of workstations, but there was no mention of high-performance networks. Although we know of a working version of MPI for Fiberchannel, the availability of MPI for high-speed networks is probably not yet significant, as gauged by the applications we have surveyed. We expect that in the future MPI will nonetheless have a wide audience on high-performance clusters.

5.3 NEEDED IMPROVEMENTS TO MPI

Respondents spoke to the issues of needed improvements to MPI. In the area of communication primitives, a common request was for a broadcast primitive to be matched up with a simple receive. Furthermore, non-blocking (or asynchronous) collective communication operations were requested. Currently, all such operations block the participating processes. The interface to derived datatypes was found unclear and more documentation and examples of such datatypes was requested. The desire to have more documentation and examples (of varying difficulty, in both C and Fortran–77) was also raised widely. Finally, one set of authors noted that MPI has limited scalability when a huge number of request objects are employed. They suggest that a more efficient mechanism for request objects be added to MPI. Finally, when respondents commented on the standards document, they indicated that it needs to be made more understandable, have more examples (in Fortran as well as C), and that other simpler explanations are needed. We noted that several such projects (such as an annotated reference manual) are fortunately underway.

6 Summary, Conclusions, Future Work

We surveyed application programmers concerning early applications in the Message-Passing Interface (MPI), and received ten responses, nine of which are described in this paper. Five of the responses were purely application projects, a sixth aimed at a class of applications, and the last four responses were generic, application-enabling libraries.

We were not able to get extensive performance feedback at this early stage. However, the variety of application efforts and target architectures is extremely positive. We expect that *many* projects will be reporting performance results during 1995 and beyond, so that this paper will serve as a supplement to such detailed case studies.

This paper is necessarily the first step of an ongoing study of the progress of these (and, hopefully, additional) key applications under MPI. As the number of applications continues to grow, we expect that a paper of this kind would continue to evolve, showing application classes that MPI impacts and describing MPI at work with other major software systems on such applications.

ACKNOWLEDGMENTS

Supported in part by the NSF Engineering Research Center for Computational Field Simulation, Mississippi State University to Anthony Skjellum, and by the United States Department of Energy under contract W-31-109-Eng-38 for William Gropp and Ewing Lusk.

We would like to thank the following researchers for contributing to the survey and to this paper: S. Pieper and R. Wiringa (Physics Division, Argonne National Laboratory), R. Pankajakshan and W. R. Briley, and J. Zhu (Mississippi State University, NSF Engineering Research Center for Computational Field Simulation), J. West, M. Stevens, L. Turcotte (Army Corps of Engineers, Vicksburg), H. W. Yau (Edinburgh Parallel Computing Centre), K. A. Cliffe and D. Brear (AEA Technology, Harwell, Great Britain), S. Lederman, A. Tsao, and T. Turnbull (Supercomputing Research Center), C. Bischof (Argonne National Laboratory), X. Sun (Duke University), J. A. Scott (Atlas Centre, Harwell, Great Britain), J. Salmon (California Institute of Technology), M. S. Warren (Los Alamos National Laboratory), D. J. Edelson (Northeast Parallel Architectures Center, Syracuse University), J. C. Hardwick (Carnegie Mellon University), and S. Ball (Australian National University).

We also wish to thank the members of the MPI Forum as a whole, for their efforts in creating MPI.

BIOGRAPHIES

Anthony Skjellum is an Assistant Professor of Computer Science at Mississippi State University and the NSF Engineering Research Center for Computational Field Simulation at Mississippi State. After receiving his Ph.D. in chemical engineering and computer science (minor) at the California Institute of Technology in 1990, he first worked at the Lawrence Livermore National Laboratory in scalable parallel libraries and message-passing paradigms for multicomputers. Since 1993, he has worked at Mississippi State; his projects include work on an implementation of MPI (with Lusk and Gropp), scalable libraries, and standardized high performance portable models for parallel machines.

Ewing Lusk is a senior computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory and Scientific Director of Argonne's Advanced Computing Research Facility. After receiving his Ph.D. in mathematics at the University of Maryland in 1970, he served first in the Mathematics Department and later in the Computer Science Department at Northern Illinois University before joining the Automated Reasoning group at Argonne in 1982. His research interests are in automated theorem proving, logic programming, and parallel computing. His current projects include an implementation of the MPI Message-Passing Standard and research into programming models for parallel architectures.

William Gropp is a computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory and Deputy Scientific Director of Argonne's High Performance Computing Research Facility. After receiving his Ph.D. in computer science from Stanford University in 1982, he held the positions of assistant (1982–1988) and associate (1988–1990) professor in the Computer Science Department of Yale University. In 1990, he joined the Numerical Analysis group at Argonne. His research interests are in adaptive methods for PDEs, software for scientific computing, and parallel computing. His current projects include an implementation of the MPI Message-Passing Standard, domain decomposition techniques for solving PDEs, and research into programming models for parallel architectures.

REFERENCES

Bischof, C., Huss-Lederman, S., Sun, X., Tsao, A., and Turnbull, T. 1994. Parallel performance of a symmetric eigensolver based on the invariant subspace decomposition approach. *Proc. Scalable High Performance Computing Conference 1994*.

Bischof, C., Huss-Lederman, S., Sun, X., Tsao, A., and Turnbull, T. 1995. A case study of mpi: Portable and efficient libraries. *Proc. Seventh SIAM Conference on Parallel Processing for Scientific Computing*.

Blelloch, G. E., Chatterjee, S., Hardwick, J. C., Sipelstein, J., and Zagha, M. 1994. Implementation of a portable nested data-parallel language. *J. Parallel and Distributed Computing* 21(1):4–14.

Burns, G., Daoud, R., and Vaigl, J. 1994. Lam: An open cluster environment for mpi. URL ftp: //tbag.osc.edu/pub/lam/lam-papers.tar.Z.

Cameron, G., Smith, M., White, M., Wylie, B. J. N., and McArthur, D. 1994. The PARAMICS Parallel Microscopic Traffic Simulator. *UK IT Forum Conference Digest.*

Cameron, G., Wylie, B. J. N., and McArthur, D. 1994. PARAMICS: Moving Vehicles on the Connection Machine. *Supercomputing '94:* 291–300.

Doss, N., Gropp, W., Lusk, E., and Skjellum, A. 1993. A model implementation of MPI. Argonne National Laboratory.

Franke, H., Hochschild, P., Pattnaik, P., and Snir, M. 1993. An efficient implementation of MPI. frankeh@watson.ibm.com.

Gropp, W., Lusk, E., and Skjellum, A. 1994. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press.

Hardwick, J. C. 1994. Porting a vector library: a comparison of mpi, paris, cmmd and pvm. CMU-CS-94-200. Carnegie Mellon University, School of Computer Science.

Lorensen, W., and Cline, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics* 21(4):163–169 (Proc. SIGGRAPH 1987).

MPI Forum 1994a. Document for a standard message-passing interface. CS-93-214 (revised). University of Tennessee.

MPI Forum 1994b. MPI: A message-passing interface standard. CS-94-230. Computer Science Department. University of Tennessee, Knoxville. *Intern. J Supercomputer Applications* 8(3/4):159–416.

Salmon, J. K., Winckelmans, G. S., and Warren, M. S. 1994. Fast parallel tree codes for gravitational and fluid dynamical N-body problems. *J. Supercomputer Appl.* 8(2):129–142; ftp/nbody/ijsa.ps.Z@sampson.ccsf.caltech.edu.

Skjellum, A., Doss, N. E., and Viswanathan, K. 1994. Inter-communicator Extensions to MPI in the MPIX (MPI eXtension) Library (submitted to ICAE Journal: Special Issue on Distributed Computing).

Taylor, L. K., Busby, J. A., Jiang, M. Y., Arabshahi, A., Sreenivas, K., and Whitfield, D. L. 1993. Time accurate incompressible Navier-Stokes simulation of the flapping foil experiment. Presented at the Sixth International Conference on Numerical Ship Hydrodynamics, Iowa City, Iowa.

Taylor, L. K., and Whitfield, D. L. 1991. Unsteady three-dimensional incompressible Euler and Navier-Stokes solver for stationary and dynamic grids. AIAA 91-1650. Presented at the AIAA 22nd Fluid Dynamics, Plasma Dynamics and Lasers Conference, Honolulu, Hawaii.

Warren, M. S., and Salmon, J. K. 1993. A parallel hashed oct-tree N-body algorithm. In *Supercomputing '93* Los Alamitos. pp. 12–21. IEEE Computer Society.

West, J. E., Stephens, M. M., and Turcotte, L. H. 1994a. Adaptation of volume visualization techniques to MIMD architectures using MPI. In *Proc. IEEE Scalable Parallel Libraries Conference*, edited by A. Skjellum and D. Reese. Los Alamitos, CA: IEEE Computer Society Press.

Westover, L. 1990. Footprint evaluation for volume rendering. (Proc. SIGGRAPH '90) *Computer Graphics* 24(4):367–376.

Whitfield, D. L., and Taylor, L. K. 1991. Discretized Newton-relaxation solution of high resolution flux-difference split schemes. AIAA-91-1539. Presented at the AIAA 10th Computational Fluid Dynamics Conference, Honolulu, Hawaii.

Wylie, B. J. N., Cameron, G., White, M., Smith, M., and McArthur, D. 1993. PARAMICS: Parallel Microscopic Traffic Simulator. In *Proc. Second European Connection Machine Users Confeence.*