MPI-GLUE: Interoperable High-Performance MPI Combining Different Vendor's MPI Worlds

Rolf Rabenseifner

Rechenzentrum der Universität Stuttgart, Allmandring 30, D-70550 Stuttgart,
rabenseifner@rus.uni-stuttgart.de,
http://www.hlrs.de/people/rabenseifner/

Abstract. Several metacomputing projects try to implement MPI for homogeneous and heterogeneous clusters of parallel systems. MPI-GLUE is the first approach which exports nearly full MPI 1.1 to the user's application without losing the efficiency of the vendors' MPI implementations. Inside of each MPP or PVP system the vendor's MPI implementation is used. Between the parallel systems a slightly modified TCP-based MPICH is used, i.e. MPI-GLUE is a layer that combines different vendors' MPIs by using MPICH as a global communication layer. Major design decisions within MPI-GLUE and other metacomputing MPI libraries (PACX-MPI, PVMPI, Globus and PLUS) and their implications for the programming model are compared. The design principles are explained in detail.

1 Introduction

The development of large scale parallel applications for clusters of high-end MPP or PVP systems requires efficient and standardized programming models. The message passing interface MPI [9] is one solution. Several metacomputing projects try to solve the problem that mostly different vendors' MPI implementations are not interoperable. MPI-GLUE is the first approach that exports nearly full MPI 1.1 without losing the efficiency of the vendors' MPI implementations inside of each parallel system.

2 Architecture of MPI-GLUE

MPI-GLUE is a library exporting the standard MPI 1.1 to parallel applications on clusters of parallel systems. MPI-GLUE imports the native MPI library of the system's vendor for the communication inside of a parallel system. Parallel systems in this sense are MPP or PVP systems, but also homogeneous workstation clusters. For the communication between such parallel systems MPI-GLUE uses a portable TCP-based MPI library (e.g. MPICH). In MPI-GLUE the native MPI library is also used by the portable MPI to transfer its byte messages inside of each parallel system. This design allows any homogeneous and heterogeneous combination of any number of parallel or nonparallel systems. The details are shown in Fig. 1.

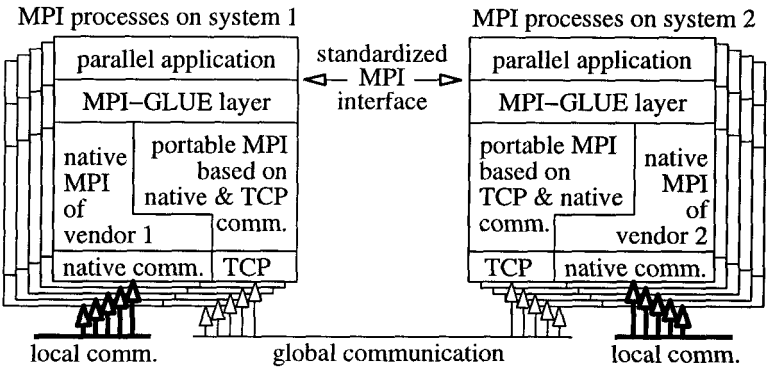

Fig. 1. Software architecture

3 Related Works

The **PACX-MPI** [1] project at the computing center of the University of Stuttgart has a similar approach as MPI-GLUE. At Supercomputing '97 PACX-MPI was used to demonstrate that MPI-based metacomputing can solve large scale problems. Two CRAY T3E with 512 processors each were combined to run a flow solver (URANUS) which predicts aerodynamic forces and high temperatures effecting on space vehicles when they reenter Earth's atmosphere [2]. Furthermore a new world record in simulating an FCC crystal with 1.399.440.00 atoms has been reached on that cluster [11]. PACX-MPI uses a special module for the TCP communication in contrast to the portable MPI in MPI-GLUE. The functionality of PACX-MPI is limited by this module to a small subset of MPI 1.1. The TCP communication does not directly exchange messages between the MPI processes. On each parallel system two special nodes are used as concentrators for incoming and outgoing TCP messages. PACX-MPI can compress all TCP messages to enlarge bandwidth.

PVMPI [4] has been developed at UTK, ORNL and ICL, and couples several MPI worlds by using PVM based bridges. The user interface does not meet the MPI 1.1 standard, but is similar to a restricted subset of the functionality of MPI_Comm_join in MPI 2.0 [10]. It is impossible to merge the bridge's intercommunicator into a global intra-communicator over all processes. Therefore, one can not execute existing MPI applications with PVMPI on a cluster of parallel systems.

In the **PLUS** [3] project at PC^2 at the University of Paderborn a bridge has been developed too. With this bridge MPI and PVM worlds can be combined. Inefficient TCP implementations of some vendors have been substituted by an own protocol based on UDP. It is optimized for wide area networks. As in PACX-MPI the communication is concentrated by a daemon process on each parallel system. The interface has the same restrictions as described for PVMPI, moreover, only a restricted subset of datatypes is valid. PLUS is part of the MOL (Metacomputer Online) project.

In the **Globus** [5] project and in the **I-WAY** [7] project MPICH [8] is used on top of the multi-protocol communication library NEXUS [6]. NEXUS is designed as a basic module for communication libraries like MPI. On the parallel system itself MPICH is used instead of the vendor's MPI library.

4 Design Decisions and Related Problems

Several decisions must be taken in the design of an MPI library usable for heterogeneous metacomputing. Major choices and the decisions in the mentioned projects are viewed in this section.

The glue approach versus the portable multi-protocol MPI implementation: The goals of high efficiency and full MPI functionality can be achieved by the following alternatives (Fig. 2): (a) The glue layer exports an MPI interface to the application and imports the native MPIs in each parallel system and a global communication module which allows communication between parallel systems. This gerneral glue approach is used in MPI-GLUE, PACX-MPI, PVMPI and PLUS. (b) In the portable multi-protocol MPI approach (used in Globus) each parallel system has its own local communication module. A common global communication module enables the communication between the parallel systems. The MPI library uses both modules. In this approach the global communication module is usually very small in comparison with the glue approach: it does only allow byte message transfers. The main disadvantage of this approach is that the performance of the native MPI library can not be used.

Global world versus dynamically created bridges: The standards MPI 1.1 and MPI 2.0 imply two metacomputing alternatives: (a) MPI_COMM_WORLD collects *all* processes (in MPI-GLUE, PACX-MPI and Globus) or (b) each partition of the cluster has a separate MPI_COMM_WORLD and inter-communicators connect the partitions. (doesn't conform to MPI 1.1, used in PVMPI and PLUS).

Global communication using an own module versus importing a portable MPI: The global communication module in the glue approach can be (a) a portable TCP-based MPI implementation (in MPI-GLUE), or (b) a special TCP-based module (in PACX-MPI, PVMPI and PLUS). With (a) its possible to implement the full MPI 1.1 standard without reimplementing most of the MPI's functionality inside of this special module.

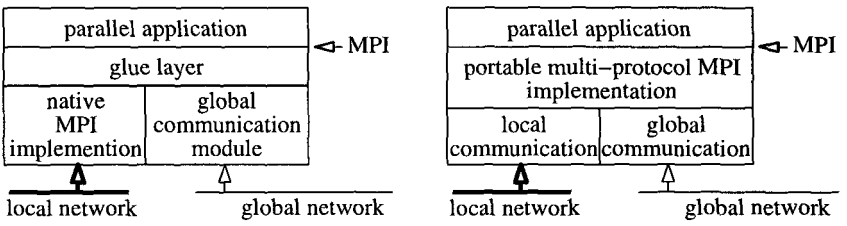

Fig. 2. The glue approach (left) and the multi-protocol MPI approach (right)

Using daemons or not using daemons: The global communication can be (a) concentrated by daemons or can be (b) done directly. In the *daemon-based approach* (a) the application processes send their messages on the local network (e.g. by using the native MPI) to a daemon, that transfers them on the global network (e.g. with TCP) to the daemon of the target system. From there the messages are received by the application processes within the local network of the target system. In the *direct-communication approach* (b) the work of the daemons is done by the operating system and each node communicates directly via the global network. With the daemon-based approach the application can fully parallelize computation and communication. The application is blocked only by native MPI latencies although in the background slow global communication is executed. Currently only the direct-communication approach is implemented in MPI-GLUE, but it would be no problem to add the daemon-based approach. A detailed discussion can be found in [12].

The first progress problem: All testing and blocking MPI calls that have to look at processes connected by the *local* network *and* at other processes connected by the *global* network should take into account that the latencies for probing the local and global network are different. On clusters of MPPs the ratio can be 1:100. Examples are a call to MPI_Recv with source=MPI_ANY_SOURCE, or a call to MPI_Waitany with requests handling local *and* global communication. More a workaround than a solution are asymmetric polling strategies: if polling is necessary then only in the n^{th} round the central polling routine will inquire the state of the global interface, while in each round the local interface will be examined. n is given by the ratio mentioned above. Without this trick the latency for local communication may be expanded to the latency of the global communication. But with this trick the latency for global communication may be in the worst case twice the latency of the underlaying communication routine. This first progress problem arises in all systems that use no daemons for the global communication (MPI-GLUE, PVMPI configured without daemons, and Globus). In MPI-GLUE the asymmetric polling strategy has been implemented. In Globus it is possible to specify different polling intervals for each protocol.

The second progress problem: The restriction in MPI 1.1, page 12, lines 24ff ("This document specifies the behavior of a parallel program assuming that only MPI calls are used for communication") allows that MPI implementations make progress on non-blocking and buffered operations *only inside* of MPI routines, e.g. in MPI_Wait... / _Test... / _Iprobe or _Finalize. This is a weak interpretation of the progress rule in MPI 1.1, page 44, lines 41ff. The rationale in MPI 2.0, page 142, lines 14-29 explicitely mentions this interpretation. This allows that a native MPI makes no progress, while the portable MPI is blocked by waiting for a message on the TCP link, and vice versa.

Therefore, in the glue approach the problem arises that the glue layer must not implement anything by blocking or testing only within the native MPI without giving the global communication module a chance to make progress, and vice versa. The only *hard problem* is the combination of using collective routines of the native MPI (because there isn't any non-blocking alternative) *and*

using sends in the global MPI *and* these sends are buffered at the sender *and* the sender is blocked in the native collective operation. This problem can be solved in daemon-based systems by sending all messages (with a destination in another parallel system) immediately to the daemons and buffering them there (implemented in PACX-MPI and PLUS). It can also be solved by modifying the central probe and dispatching routine inside the native MPI.

The analogous problem (how to give the native MPI a chance to make progress, while the glue layer blocks in a global collective routine) usually only exists, if one uses an existing MPI as a global communication module, but in MPI-GLUE this problem has been solved, because the global MPI itself is a multi-protocol implementation which uses the local MPI as one protocol stack.

5 Design Details

The notation **"MPI"** is used for the MPI interface exported by this glue layer and for the glue layer software.
The notation **"MPX"** is used for the imported TCP-based MPI-1.1 library in which all names are modified and begin with "MPX_" instead of "MPI_". MPX allows the exchange of messages between all processes.
The notation **"MPL"** is used for the native (local) MPI 1.1 library. All names are modified ("MPL_" instead of "MPI_"). All MPL_COMM_WORLDs are disjoint and in the current version each process is member of exactly one MPL_COMM_WORLD (that may have any size≥1).

Details are given in the table below and in [12]. In the table **"X"** and **"L"** are abbreviations for MPX and MPL.

MPI constant, handle, routine	impl.ed by using	remarks
ranks	X	i.e. the MPI ranks are equal to the MPX ranks.
group handling	X	i.e. group handles and handling are impl. with MPX.
communicators	L and X	and *roots-communicator*, see Sec. *Opt. of collective op.*
datatypes	L and X	and a mapping of the L ranks to the X ranks and vice versa.
requests	L or X or delayed	MPI_IRECVs with MPI_ANY_SOURCE and communicators spawning more than one MPL region store all arguments of the IRECV call into the handle's structure. The real RECV is delayed until the test or wait calls.
sends, receives	L or X	(receives only with known source partition).
receives with ANY_SOURCE:		
- blocking	L and X	with asymmetric polling strategy.
- nonblocking	GLUE	postponed until the wait and test routines, see MPI requests above.
wait and test	L or X or L&X.	
MPI_Testall		Testing inside partial areas of the communicator may complete request handles, but the interface definition requires that only none or all requests are completed. Therefore (only for Testall), requests may be MPL/MPX-finished, but not MPI-finished.

The MPI process start and MPI_Init is mapped to the MPL process start, MPL_Init, MPX process start and MPX_Init. The user starts the application processes on the first parallel system with its MPL process start facility. MPI_Init first calls MPL_Init and then MPX_Init. The latter one is modified to reflect that the parallel partition is already started by MPL. Based on special arguments at the end of the argument list and based on the MPL ranks the first node recognizes inside of MPX_Init its role as MPICH's *big master*. It interactively starts the other partitions on the other parallel systems or uses a *remote shell* method (e.g. `rsh`) to invoke the MPL process start facility there. There as well the MPI_Init routine first invokes MPL_Init and then MPX_Init. Finally MPI_Init initializes the predefined handles. MPI_Finalize is implemented in the reverse order.

Optimization of collective operations: It is desirable that metacomputing applications do not synchronize processes over the TCP links. This implies that they should not call collective operations on a global communicators, especially those operations which make a barrier synchronization (barrier, allreduce, reduce_scatter, allgather(v), alltoall(v)). The following schemes help to minimize the latencies if the application still uses collective operations over global communicators. In general a collective routine is mapped to the corresponding routine of the native MPI *if the communicator is completely inside of one local MPL world.* To optimize MPI_Barrier, MPI_Bcast, MPI_Reduce and MPI_Allreduce *if the communicator belongs to more than one MPL world*, for each MPI-communicator, there is an additional internal *roots-communicator* that combines the first process of any underlying MPL-communicator by using MPX, e.g. MPI_Bcast is implemented by a call to native Bcast in each underlying MPL-communicator and one call to the global Bcast in the roots-communicator.

6 Status, Future Plans, and Acknowledgments

MPI-GLUE implements the full MPI 1.1 standard except some functionalities. Some restrictions result from the glue approach: (a) It is impossible to allow an unlimited number of user-defined operations. (b) Messages must be and need to be received with datatype MPI_PACKED if and only if they are sent with MPI_PACKED. Some other (hopefully minor) functionalities are still waiting to be implemented. A detailed list can be found in [12]. MPI-GLUE is portable. Current test platforms are SGI, CRAY T3E and Intel Paragon. Possible future extensions can be the implementation of MPI 2.0 functionalities, optimization of further collective routines, and integrating a daemon-based communication module inside the used portable MPI.

I would like to thank the members of the MPI-2 Forum and of the PACX-MPI project as well as the developers of MPICH who have helped me in the discussion which led to the present design and implementation of MPI-GLUE. And I would like to thank Oliver Hofmann for implementing a first prototype in his diploma thesis, Christoph Grunwald for implementing some routines during his practical course, especially some wait, test and topology functions, and Matthias Müller for testing MPI-GLUE with his P3T-DSMC application.

7 Summary

MPI-GLUE achieves interoperability for the message passing interface MPI. For this it combines existing vendors' MPI implementations losing neither full MPI 1.1 functionality nor the vendors' MPIs' efficiency. MPI-GLUE targets existing MPI applications that need clusters of MPPs or PVPs to solve large-scale problems. MPI-GLUE enables metacomputing by seamlessly expanding MPI 1.1 to any cluster of parallel systems. It exports a single virtual parallel MPI machine to the application. It combines all processes in one MPI_COMM_WORLD. All local communication is done by the local vendor's MPI and all global communication is implemented with MPICH directly, without using deamons. The design has only two unavoidable restrictions which concern user defined reduce operations and packed messages.

References

1. Beisel, T., Gabriel, E., Resch, M.: An Extension to MPI for Distributed Computing on MPPs. In Marian Bubak, Jack Dongarra, Jerzy Wasniewski (Eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, LNCS, Springer, 1997, 75-83.
2. T. Bönisch, R. Rühle: Portable Parallelization of a 3-D Flow-Solver, Parallel Comp. Fluid Dynamics '97, Elsevier, Amsterdam, to appear.
3. Brune, M., Gehring, J., Reinefeld, A.: A Lightweight Communication Interface for Parallel Programming Environments. In Proc. High-Performance Computing and Networking HPCN'97, LNCS, Springer, 1997.
4. Fagg, G.E., Dongarra, J.J.: PVMPI: An Integration of the PVM and MPI Systems. Department of Computer Science Technical Report CS-96-328, University of Tennessee, Knoxville, May, 1996.
5. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications (to appear).
6. Foster, I., Geisler, J., Kesselman, C., Tuecke, S.: Managing Multiple Communication Methods in High-Performance Networked Computing Systems. J. Parallel and Distributed Computing, 40:35-48, 1997.
7. Foster, I., Geisler, J., Tuecke, S.: MPI on the I-WAY: A Wide-Area, Multimethod Implementation of the Message Passing Interface. Proc. 1996 MPI Developers Conference, 10-17, 1996.
8. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Paralle Computing, 22, 1996.
9. MPI: A Message-Passing Interface Standard. Message Passing Interface Forum, June 12, 1995.
10. MPI-2: Extensions to the Message-Passing Interface. Message Passing Interface Forum, July 18, 1997.
11. Müller, M.: Weltrekorde durch Metacomputing. BI 11+12, Regionales Rechenzentrum Universität Stuttgart, 1997.
12. Rabenseifner, R.: MPI-GLUE: Interoperable high-performance MPI combining different vendor's MPI worlds. Tech. rep., Computing Center University Stuttgart. `http://www.hlrs.de/people/rabenseifner/publ/mpi_glue_tr_apr98.ps.Z`