Heterogeneous MPI Application Interoperation and Process Management under *PVMPI* *

Graham E. Fagg[1], Jack J. Dongarra[1,2] and Al Geist[2]

[1] Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
[2] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6367

Abstract. Presently, different vendors' MPI implementations cannot interoperate directly with each other. As a result, performance of distributed computing across different vendors' machines requires use of a single MPI implementation, such as MPICH. This solution may be sub-optimal since it cannot utilize the vendors' own optimized MPI implementations. *PVMPI*, a software package currently under development at the University of Tennessee, provides the needed interoperability between different vendors' optimized MPI implementations. As the name suggests *PVMPI* is a powerful combination of the proven and widely ported Parallel Virtual Machine (PVM) system and MPI. *PVMPI* is transparent to MPI applications thus allowing intercommunication via all the MPI point-to-point calls. Additionally, *PVMPI* allows flexible control over MPI applications by providing access to all the process control and resource control functions available in the PVM virtual machine.

1 Introduction

The past several years have seen numerous efforts to address the deficiencies of the different message passing systems and to introduce a single standard for such systems. These efforts culminated in the first Message Passing Interface (MPI) standard, introduced in June 1994 [15]. Within a year, various implementations of MPI were available, including both commercial and public domain systems.

One of MPI's prime goals was to produce a system that would allow manufacturers of high-performance massively parallel processing (MPPs) computers to provide highly optimized and efficient implementations. In contrast, systems such as PVM [1] were designed for clusters of computers, with the primary goals of portability, and ease-of-use. These have been achieved with little loss of performance [4] and with greater flexibility than the native communications system.

* This work was supported in part by the NSF under grant ASC-9214149, the Mathematical, Information and Computer Sciences subprogram of the Office of Energy Research, DOE, under Contract DE-AC05-84OR21400, ,Rice University and The State of Tennessee.

The aim of *PVMPI* is to interface the flexible process and virtual machine control from the PVM system with several optimized MPI communication systems thus allowing MPI applications the ability to interoperate transparently across multiple heterogeneous hosts.

2 Virtual Machine Resource and Process Control

The PVM virtual machine is defined to be a dynamic collection of parallel and serial hosts. With the exception of one host in the PVM virtual machine, any number of hosts can join, leave, or fail without affecting the rest of the virtual machine. In addition, the PVM resource control API allows the user to add or delete hosts, check that a host is responding, shut down the virtual machine or be notified by a user-level message that a host has been added or deleted (intentionally or not).

The PVM virtual machine is very flexible in its process control capabilities. It can start serial, or parallel processes that may or may not be PVM applications. For example, PVM can spawn an MPI application as easily as it can spawn a PVM application. The PVM process control API allows any process to join or leave the virtual machine, start new processes by using a number of different selection criteria (including external schedulers, resource managers and/or taskers), signal or kill a process, test to check that a process is responding and, notify an arbitrary process if another disconnects from the PVM system.

In addition to the above virtual machine control functions, PVM provides plug-in interfaces for expanding its resource and process control capabilities. This extendibility has encouraged many projects to use PVM in different distributed computing environments such as Mist [14], dedicated schedulers [10], load balancers and process migration tools [5, 16].

2.1 PVM Group Services

PVM provides the ability to group processes within the virtual machine. Groups are identified by a character string name. Processes can join and leave any number of groups at any time, thus making group membership completely dynamic. Processes are allocated instance numbers when they join a group, in the order of membership. The first join operation creates the group, and the group is destroyed when the membership falls to zero. Groups may have gaps in their membership as processes leave out of order. To improve performance, PVM allows group membership to be frozen by caching group details locally. Fully dynamic group caching is also available[11][12]. Many users only use the PVM group functions as a convenient naming/binding service.

3 MPI Communicators

Although the MPI standard does not specify how processes are started, it does dictate how MPI processes enroll into the MPI system. All MPI processes join the

MPI system by calling MPI_Init and leave it by calling MPI_Finalize. Calling MPI_Init twice causes undefined behavior. Processes in MPI are arranged in rank order, from 0 to N-1, where N is the number of processes in a group. These process groups define the scope for all collective operations within that group. Communicators consist of a process group, context, topology information and local attribute caching. All MPI communications can only occur within a communicator.

Once all the expected MPI processes have started a common communicator is created by the system for them called MPI_COMM_WORLD. Communications between processes within the same communicator or group are referred to as *intra-communicator communications*. Communications between disjoint groups are *inter-communicator communications*. The formation of an inter-communicator requires two separate (non overlapping) groups and a common communicator between the leaders of each group, as shown in Figure 1.

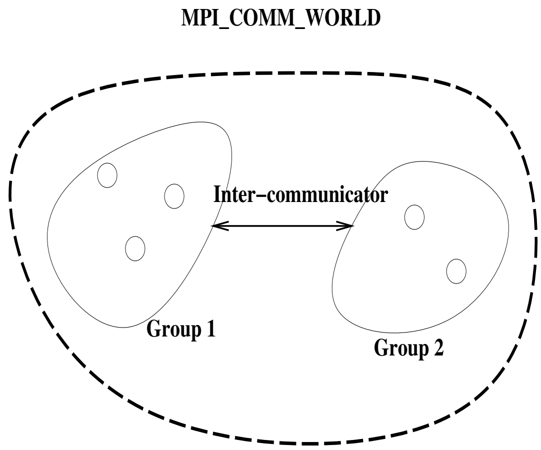

Fig. 1. Inter-communicator formed inside a single MPI_COMM_WORLD

The MPI-1 standard does not provide a way to create an inter-communicator between two separately initiated MPI applications since no global communicator exists between them. The scope of each application is limited by its own MPI_COMM_WORLD which by its nature is distinct from any other applications' MPI_COMM_WORLD. Since all internal details are hidden from the user and MPI communicators have relevance only within a particular run-time instance, MPI-1 implementations cannot inter-operate.

4 Related Work

Although several MPI implementations are built upon other established message-passing libraries such as Chameleon-based MPICH [7], LAM [3] and Unify [6], none allow true inter-operation between separate MPI applications across different MPI implementations.

LAM 6.X does allow some limited interaction between LAM only applications using a subset of functions from the dynamic process chapter of the proposed MPI-2 standards document.

Unify system was originally proposed to *unify* or mate together the PVM and new MPI APIs. The intention was to enable users to take current PVM applications and slowly migrate toward complete MPI applications, without having to make the complete conceptual jump from one system to the other. The project never reached full maturity although it did address the difficulty of mapping identifiers between the PVM and MPI domains which it solved using additional function calls.

The only project known to the authors that attempts to directly interconnect MPI applications in a way similar to *PVMPI* is currently under way at the Computer Centre of the Rechenzentrum Universitaet in Stuttgart[2]. This project attempts to interconnect pairs of MPPs via specialist processes that use standard TCP/IP for communications.

5 The *PVMPI* System

We developed a prototype system[9] to study the issues of interconnecting MPI and PVM. Three separate issues were addressed:

1. mapping identifiers and managing MPI and PVM IDs
2. transparent MPI message passing
3. start-up facilities and process management

5.1 Mapping Identifiers

A process in an MPI application is identified by a tuple pair either {process group, rank} or {communicator, rank}. PVM provides similar functionality through use of the group library. The PVM tuple is {group name, instance}. *PVMPI* provides address mapping from the MPI tuple space to the PVM tuple space and vice versa. An initial prototype version of *PVMPI*[8] used such a system without any further translation (or hiding of mixed identifiers).

The association of this tuple pair is achieved by registering each MPI process into a PVM group by a user level function call. A matching dis-associate or leave call is also provided.

The functions are available in both C and Fortran bindings:

```
info = PVMPI_Register(char *group, MPI_Comm comm, int *handle);
info = PVMPI_Leave(char *group);

call pvmpi_register( group, comm, handle, info )
call pvmpi_leave ( group, info )
```

Both register and leave functions are collective and blocking: all processes in the specified MPI communicator have to participate. The PVMPI_Leave command is used to clean up MPI data structures and to leave the PVM system in an orderly way if required.

Processes can register in multiple groups, although currently separate applications cannot register into a single group with this call (i.e. take the same named group). The register call takes each member of the communicator and makes it join a named PVM group so that its instance number within that group matches its MPI rank. Since any two MPI applications may be executing on different systems using different implementations of MPI (or even different instances of the same version), the communicator usually has no meaning outside of any application callable library. The PVM group server, however, can be used to resolve identity when the groups names are unique.

Once the application has registered, an external process can access it by using that process' group name and instance via the library calls pvm_gettid and pvm_getinst. When the groups have been fully formed, they are frozen and all their details are cached locally to reduce system over-head.

5.2 Transparent Messaging

The mixing of MPI and PVM group calls requires the understanding of two different message passing systems, their APIs, semantics and data formats. A better solution is to transparently provide interoperability of MPI application by utilizing only the MPI API.

As previously stated, MPI uses communicators to identify message universes, and not PVM group names or TIDs. Thus the *PVMPI* could not allow users to utilize the original MPI calls for inter-application communication. The solution is to allow the creation of virtual communicators that map either onto PVM and hence remote applications or onto real MPI intra-communicators for local communication.

In order to provide transparency and handle all possible uses of communicators, all MPI routines using communicators were re-implemented using MPIs profiling interface. This interface allows user library calls to be intercepted on a selective bases so that debugging and profiling tools can be linked into applications without any source code changes.

Creating dual role communicators within MPI would require altering MPI's low level structure. As this was not feasible, an alternative approach was taken. *PVMPI* maintains its own concept of a communicator using a hash table to store the actual communication parameters. As communicators in MPI are opaque data structures this behavior has no impact on end user code. Thus *PVMPI* communicator usage is completely transparent as shown in figure 2.

Intra and inter communicator communications within a single application (MPI_COMM_WORLD) proceeds as normal, while inter-application communication proceed by the use of a *PVMPI* inter-communicator formed by using the PVMPI_Intercomm_create function:

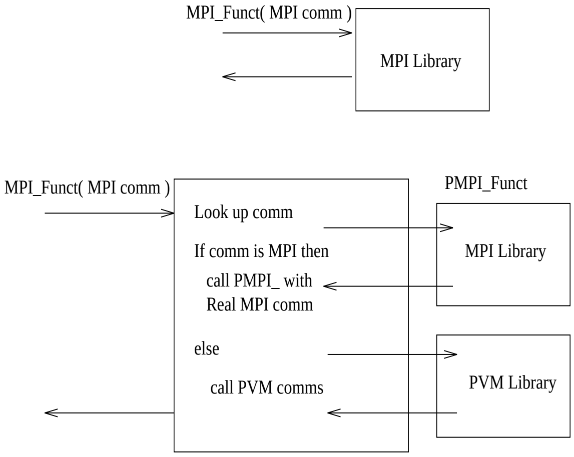

Fig. 2. MPI profiling interface controlling communicator translation.

```
info = PVMPI_Intercomm_create (int handle, char *gname, MPI_Comm *intercom);

call pvmpi_intercomm_create (handle, gname, intercom, info)
```

This function is almost identical to the normal MPI inter-communicator create call except that it takes a *handle* from the register function instead of a communicator to identify the local group, and a registered name for the remote group. The handle is used to differentiated between local groups registered under multiple names.

The default call is blocking and collective, although a non-blocking version has been implemented that can time-out or warn if the requested remote group has attempted to start and then failed, so that appropriate action can be taken to aid fault tolerance.

PVMPI inter-communicators are freed using the typical MPI function calls. They can be formed, destroyed and recreated without restriction. Once formed, they can be used exactly the same as a normal MPI inter-communicator except in the present version of *PVMPI* there is a restriction that they cannot be used in the formation of any new communicators.

PVMPI inter-communicators allow the full range of point-to-point message passing calls inside MPI. Also supported is a number of data formating and (un)packing options, including user derived data types (i.e. mixed striding and formats). Receive operations across inter-communicators relies upon adequate buffering at the receiving end, in-line with normal PVM operation.

5.3 Low-level Start-up Facilities

The spawning of MPI jobs from PVM requires different procedures depending upon the target system and the MPI implementation involved. The situation is complicated by the desire to avoid adding many additional spawn calls (the current intention of the MPI-2 forum). Instead, a number of different MPI implementation specific taskers have been developed that intercept the internal PVM spawn messages and then correctly initiate the MPI applications as required.

5.4 Process Management under a General Resource Manager

The PVM GRM[10] can be used with specialized *PVMPI* taskers to manage MPI applications in an efficient and simple manner. This provides improved performance [13] and better flexibility than that of a simple host file utilized by most MPIRUN systems.

When a user's spawn request is issued it is intercepted by the GRM and an attempt is made to optimize the placement of tasks upon available hosts. If the placement is specialized then appropriate taskers are used. Figure 3 shows a system with three clusters of machines: one each for MPICH, LAM and general purpose jobs. In this figure the start request causes two MPICH nodes to be selected by the GRM, then the actual processes are started by the MPICH tasker.

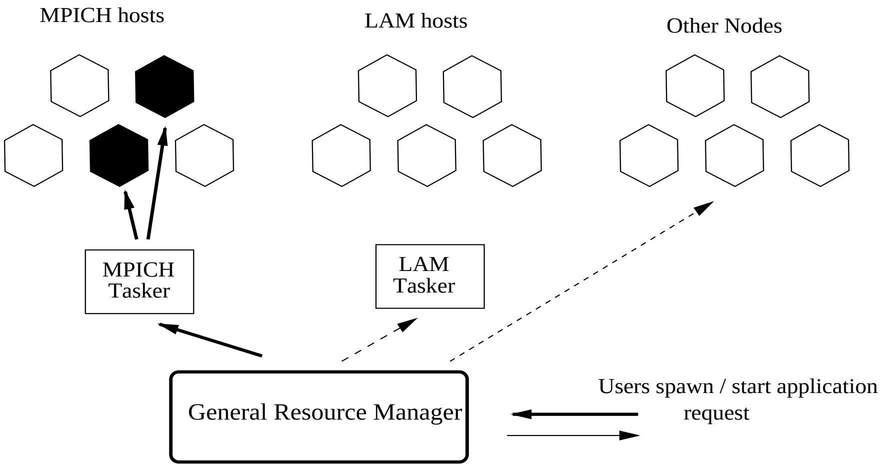

Fig. 3. General Resource Manager and Taskers handling process management

6 Conclusions

The *PVMPI* system solves the lack of interoperability between MPI-1 implementations. It allows the user to run applications across different hardware systems, while still utilizing the vendors' optimized MPI implementations on each system. *PVMPI* usage is transparent to the end user and it's usage requires only three additional calls (PVMPI_register, PVMPI_leave and PVMPI_Intercom_create). Additionally, it provides flexible process management and assists in efficient use of networked resources.

References

1. A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Manchek, and V. S. Sunderam. Heterogeneous Network Computing. *Sixth SIAM Conference on Parallel Processing*, 1993.
2. Thomas Beisel. "Ein effizientes Message-Passing-Interface (MPI) fuer HiPPI", Diploma thesis, University of Stuttgart, 1996.
3. Greg Burns, Raja Daoud and James Vaigl. LAM: An Open Cluster Environment for MPI. Technical report, Ohio Supercomputer Center, Columbus, Ohio, 1994.
4. Henri Casanova, Jack Dongarra and Weicheng Jiang. The Performance of PVM on MPP Systems. Department of Computer Science Technical Report CS-95-301. University of Tennessee at Knoxville, Knoxville, TN. August 1995.
5. J. Casas, R. Konuru, S. Otto, R. Prouty, and J. Walpole. Adaptive Load Migration Systems for PVM. *Supercomputing'94 Proceedings* , pp. 390-399, IEEE Computer Society Press, 1994.
6. Fei-Chen Cheng. Unifying the MPI and PVM 3 Systems. Technical report, Department of Computer Science, Mississippi State University, May 1994.
7. Nathan Doss, William Gropp, Ewing Lusk and Anthony Skjellum. A model implementation of MPI. Technical report MCS-P393-1193, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, 1993.
8. Graham E. Fagg and Jack J. Dongarra, PVMPI: An Integration of the PVM and MPI Systems. Calculateours Paralle'les, Paris, Vol 8/2, pp. 151-166, June 1996.
9. Graham E. Fagg, Jack J. Dongarra and Al Geist, PVMPI provides Interoperability between MPI Implementations Proceedings of Eight SIAM conference on Parallel Processing March 1997
10. Graham E. Fagg, Kevin London and Jack J. Dongarra, Taskers and General Resource Manager: PVM supporting DCE Process Management, Proceeding of the third EuroPVM group meeting, Munich, Springer Verlag, October 1996.
11. G.E. Fagg, R.J. Loader, P.R. Minchinton and S.A. Williams. Improved Group Services for PVM. Proceeding of *1995 PVM Users Group Meeting*, Pittsburgh, pp.6, May 1995.
12. Graham E. Fagg, Roger J. Loader and Shirley A. Williams. Compiling for Groups. Proceeding of *EuroPVM 95*, pp. 77-82, Hermes, Paris, 1995.
13. Graham E. Fagg and Shirley A. Williams. Improved Program Performance using a cluster of Workstations. *Parallel Algorithms and Applications*, Vol 7, pp. 233-236, 1995.
14. R. Konuru, J. Casas, S. Otto, R. Prouty and J. Walpole. A User-Level Process Package for PVM. *Scalable High Performance Computing Conference*, pp. 48-55, IEEE Computer Society Press, 1994.
15. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994. Special issue on MPI.
16. Georg Stellner and Jim Pruyne. Resource Management and Checkpointing for PVM Proceeding of *EuroPVM 95*, pp. 130-136, Hermes, Paris, 1995.

This article was processed using the LaTeX macro package with LLNCS style