

# Distributed computing in a heterogeneous computing environment

Edgar Gabriel, Michael Resch, Thomas Beisel and Rainer Keller

High Performance Computing Center Stuttgart  
Parallel Computing Department  
Allmandring 30  
D-70550 Stuttgart, Germany  
gabriel@hls.de  
resch@hls.de

**Abstract.** Distributed computing is a means to overcome the limitations of single computing systems. In this paper we describe how clusters of heterogeneous supercomputers can be used to run a single application or a set of applications. We concentrate on the communication problem in such a configuration and present a software library called PACX-MPI that was developed to allow a single system image from the point of view of an MPI programmer. We describe the concepts that have been implemented for heterogeneous clusters of this type and give a description of real applications using this library.

## 1 Introduction

The simulation of very large systems often requires computational capabilities which cannot be satisfied by a single massively parallel processing system (MPP) or a parallel vector processor (PVP). A possible way to solve this problem is to couple different computational resources distributed all over the world. Although there are different definitions of metacomputing, this is how in the following we use this term.

The coupling of MPPs and/or PVPs requires a reliable and - if possible - dedicated network connection between the machines, a software which enables interoperability but also algorithms which allow to exploit the underlying power of such a metacomputer. Loosely coupled applications and applications with a clear break in the communication pattern seem to be optimal for metacomputing [1, 2]. However, it was shown, that even closely coupled applications can be adapted for metacomputing [3].

Since MPI as the message-passing standard provides no interoperability, one has to use special software to make an application run on a metacomputer. Several libraries are available that give support in this field. MPICH [4] provides a kind of interoperability, but all communication has to be done via TCP/IP. The number of TCP connections e.g. for the coupling of two Cray T3Es may be as high as  $512 \cdot 512$ . PVMPI [5] is a library that enables the coupling of already running MPI applications using PVM for the communication. This concept has

two major disadvantages: firstly the user has to change his sources for that and secondly he may only use point-to-point communication after having created an inter-communicator between the different machines. A similar library - developed at the *PC<sup>2</sup>* - is PLUS [6]. It enables the coupling of different message-passing libraries such as MPI, PVM or PARMACS. Again the user has to add some non-standard calls to the code; and again only point-to-point communication is possible.

We have recently presented a library to couple different MPPs called PACX-MPI (PARallel Computer eXtension) [7]. The major advantage of this library is that the user does not need to change his code; he may simply develop the program on a smaller test case using a single MPP and then couple different supercomputers by only linking the PACX-MPI library to his application.

In chapter two of this paper we will briefly present the concepts of PACX-MPI focussing on the major improvements and experiences during the last year. Chapter three presents some applications using PACX-MPI and the results we have achieved. Finally we give a brief outlook of the developments in the project and of ongoing metacomputing activities.

## 2 Concept of PACX-MPI

PACX-MPI is a library that enables the clustering of two (Version 2.0) or more (Version 3.0) MPPs into one single resource. This allows to use a metacomputer just like an ordinary MPP. The main goals of the PACX-MPI project have already been described in [7] and are only summarised here:

- No changes in the source code
- The programmer should have a single system image
- Use of the vendor implemented fast MPI for internal communication
- Use of a standard protocol for external communication.

### 2.1 Usage concept of PACX-MPI

To use PACX-MPI for an application, one has to compile and link the application with the PACX-MPI library. The main difference for the user is the start-up of the application. First he has to provide two additional nodes on each machine, which handle the external communication. An application that needs 1024 nodes on a T3E thus takes 514 if running on two separate T3Es. Then he has to configure a hostfile, which has to be identical on each machine. The hostfile contains the name of the machines, the number of application nodes, the used protocol for the communication with this machine and optionally the start-up command, if one wants to make use of the automatic start-up facility of PACX-MPI 3.0. Such a hostfile may look like this:

```

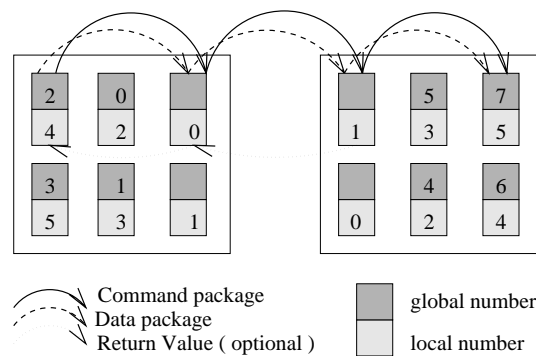
#machine nodes protocol start-up command
host1 100 tcp
host2 100 tcp (rsh host2 mpirun -np 102 ./exename)
host3 100 tcp (rsh host3 mpirun -np 102 ./exename)
host4 100 tcp (rsh host4 mpirun -np 102 ./exename)

```

## 2.2 Technical concept of PACX-MPI

To enable metacomputing, PACX-MPI redirects the MPI-calls to its own PACX-MPI library calls. For applications written in C this is done by using a macro-directive. Fortran applications first have to link with the PACX-MPI library before linking with the original MPI-library. Thus PACX-MPI is a kind of additional layer between the application and MPI.

The creation of a distributed global `MPI_COMM_WORLD` requires two numberings for each node; a local number for the `MPI_COMM_WORLD` locally established and a global one. In figure 1 the local numbers are in the lower part of the boxes and the global numbers in the upper one. The external communication is handled by two additional communication nodes, which are not considered in the global numbering. Since for the application only the global numbering is relevant these communication nodes are completely transparent. To explain their role in PACX-MPI, we describe the sequence of a point-to-point communication between global node two and global node seven. The sending node will check



**Fig. 1.** Point to point communication for a metacomputer consisting of two machines

first, whether the receiver is on the same MPP or not. If it is on the same machine, it will do a normal `MPI_Send`. If it is not, it creates a command-package, which has the same function as the message-envelope in MPI, and transfers this command-package and the data to one of the communication nodes, the so-called MPI-server. The MPI-server compresses the data and transfers them

via TCP/IP to the destination machine. There the command-package and the data are received by the so-called PACX-server, the second communication node. Data are decompressed and passed on to the destination node seven. This is done by mapping the global node number to the local one and using native vendor MPI.

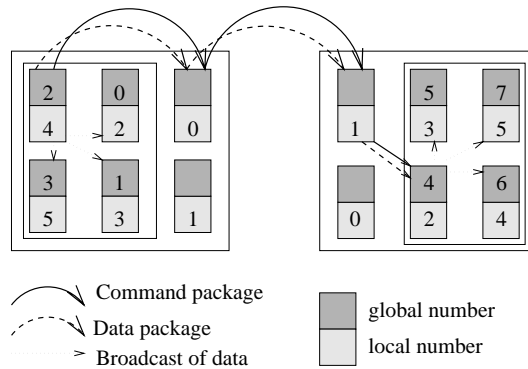
**Buffering concept of PACX-MPI** In previous PACX-MPI versions the receiver only checked whether the sending node is on the same machine or not. This is no problem for the coupling of two machines but may lead to race conditions if more than two machines are involved or if one process belongs to more than one communicator. To avoid this, message buffering on the receiving side is done. The decision was to buffer messages without matching `MPI_Recv` at the destination node rather than at the PACX-server. This allows to distribute both memory requirements and working time. In a point-to-point communication the receiving node first checks whether the message is an internal one. In case it is, it is received by directly using MPI. If it is not an internal message, the receiving node has to check whether the expected message is already in the buffer. Only if this is not the case, the message is received from the PACX-server directly.

**Data conversion in PACX-MPI** To support heterogeneous metacomputing, the new version of PACX-MPI has to do data conversion. Initially we thought of having the two communication nodes handle all data conversion. However, for the `MPI_Packed` datatype the receiver has to know exactly what the content of the message is. Therefore we decided to design the data conversion concept as follows:

- The sending node does a data conversion into an XDR-data format, if it prepares a message for another machine. For internal communication, no additional work accrues.
- The receiver converts the data from the XDR-format into its own data representation.
- For the data-type `MPI_PACKED` a data conversion to XDR-format will be done while executing `MPI_PACK`, even for internal communication.
- Because of the high overhead, data conversion can be enabled and disabled by a compiler option of PACX-MPI. This allows the optimisation of applications for homogeneous metacomputing.

### 2.3 Global Communication in PACX-MPI

In the PACX-MPI Version 2.0, some parts of the global communication were executed by the communication nodes [7]. As there are situations in which this can lead to a blocking of the application, the communication nodes are no longer involved in global operations in the current version. The sequence of a broadcast operation of node 2 to `MPI_COMM_WORLD` is shown in fig. 2. At first the root-node of the broadcast sends a command-package and a data-package to the



**Fig. 2.** A broadcast operation in PACX-MPI 3.0

MPI-server. Then a local `MPI_Bcast` is executed. Both the command-package and the data-package are transferred to the other machines. There PACX-server transfers the command and the data-package to the node with the smallest local number. This node does the local broadcast. This means that global operations are handled locally by nodes from the application part now rather than by one of the servers.

### 3 Results

Up to now, PACX-MPI is used by a small number of applications [1]. There are two applications that are developed at Stuttgart which we would like to describe here briefly.

The first metacomputer consisted of the Cray T3E 512 of the Stuttgart University and the Cray T3E 512 of the Pittsburgh Supercomputing Center. For the tests we had a dedicated network connection with a bandwidth of 2Mbits/second and a physical latency of about 70ms. Using PACX-MPI we achieved a sustained bandwidth of about 1 Mbit/s and a latency of about 75ms.

**URANUS** The first application is a Navier-Stokes Solver called URANUS (Upwind Relaxation Algorithm for Nonequilibrium flows of the University of Stuttgart), which is used to simulate the non-equilibrium flows around the re-entry vehicle in a wide altitude-velocity range [3]. The code is based on a regular grid decomposition, which leads to a very good load balancing and a simple communication pattern. URANUS was chosen for our metacomputing experiments because the simulation of re-entry vehicles requires an amount of memory that can not be provided by a single machine.

In the following we give the overall time it takes to simulate a medium size problem with 880.000 grid cells. For the tests we simulated 10 Iterations. We compared a single machine with 128 nodes and two machines with 2 times 64

Method	128 nodes using MPI	2*64 nodes using PACX-MPI
URANUS unchanged	102.4	156.7
URANUS modified	91.2	150.5
URANUS pipelined	-	116.7

**Table 1.** Comparison of timing results (sec) in metacomputing for URANUS

nodes. Obviously the unchanged code is much slower on two machines. However, the overhead of 50% is relatively small with respect to the slow network. Modification of the pre-processing does not improve the situation much. A lot more can be gained by fully asynchronous message-passing. Using so called "Message Pipelining" [3] messages are only received if available. The receiving node may continue the iteration process without having the most recent data in that case. This helped to reduce the computing time significantly. Tests for one single machine were not run because results are no longer comparable with respect to numerical convergence. Based on this final version, however, a full space vehicle configuration using more than 1 million cells was run on 760 nodes successfully during SC'97.

However, this requires a minor change in the code which shows that for closely coupled applications metacomputing can not be exploited easily without modifying the algorithm. Still the advantage for the user with PACX-MPI is, that the code remains portable.

**P3T-DSMC** The second application is P3T-DSMC. This is an object-oriented Direct Simulation Monte Carlo Code which was developed at the Institute for Computer Applications (ICA I) of Stuttgart University for general particle tracking problems [8].

Since Monte Carlo Methods are well suited for metacomputing, this application gives a very good performance on the transatlantic connection. For small number of particles the metacomputing shows some overhead. But up to 125.000 particles timings for one time step are identical. This excellent behaviour is due to two basic features of the code. First, the computation to communication ratio is becoming better if more particles are simulated per process. Second, latency can be hidden more easily if the number of particles increases.

During SC'97 this application was able to set a new world record for molecular dynamics simulating a crystal with 1.4 billion particles on two T3E's using 1024 processors. This gives hope that metacomputing can be a tool to further push the frontiers of scientific research.

In a joint effort with the Forschungszentrum Jülich GmbH at the beginning of May the new PACX-MPI Version successfully coupled three Cray T3E's for one application; the Cray T3E/900-512 at Stuttgart, a Cray T3E/900-256 and a Cray

Particles/CPU	60 nodes using MPI	2*30 nodes using PACX-MPI
1935	0.05	0.28
3906	0.1	0.31
7812	0.2	0.31
15625	0.4	0.4
31250	0.81	0.81
125000	3.27	3.3
500000	13.04	13.4

**Table 2.** Comparison of timing results (sec) in metacomputing for P3T-DSMC

T3E/600-512 at Jülich. The main difference between this metacomputer and the one used during SC'97 is that for this one Internet was used for coupling and therefore neither reliable bandwidth nor a constant latency could be achieved.

The latency between Stuttgart and Jülich was between 13 ms and 35ms with an average value of about 29 ms. Bandwidth was about 0.27 MB/second. The latency between the two T3E's of Jülich was about 7ms and the bandwidth was about 14 MB/second.

The application used was a version of P3T-DSMC which has to communicate a lot more than the previous test case does. That is why the results for this test were not as good as the previous ones. For a smaller test-suite using 34 nodes on three machines we show in the following table the overall time for a simulation:

Simulation on a single T3E	126.6 sec
Metacomputer with data-compression	163.2 sec
Metacomputer without data-compression	220.8 sec

**Table 3.** Simulation times for a P3T-DSMC test case with 3 MPP's

The table points out, that one really benefits from using the data-compression of PACX-MPI here. This behaviour was not seen when dedicated networks were used.

## 4 Outlook

We have presented new results of a library, that enables MPI-based metacomputing. Although the application test cases exhibit some overhead for metacomputing, it was shown that based on PACX-MPI we can solve problems that cannot be solved otherwise. However, metacomputing should not be understood as an

alternative to very big MPPs. It is mainly suitable for problems that cannot be solved because of lacking performance and resources such as main memory at one site.

The development of PACX-MPI will be forced in the near future, as there are requests from users for such a library. Therefore the first advancement of PACX-MPI will be to enlarge the number of supported functions, according to our application needs.

Another main direction will be the support of some parts of the MPI-2 functionality, especially support for dynamic process start. But here we will have to wait until first vendor implementations of the new standard are available.

As a protocol for external communication, only TCP/IP is supported at the moment. For long distance connections, there actually seems to be no alternative with respect to reliability. But for local networks it would be reasonable to support other network protocols like HiPPI or ATM.

## References

1. Th. Eickermann, J. Heinrichs, M. Resch, R. Stoy, R. Völpel, 'Metacomputing in Gigabit Environments: Networks, Tools and Applications' to appear in *Parallel Computing* (1998).
2. Toshiya Kimura, Hiroshi Takemiya, 'Local Area Metacomputing for Multidisciplinary Problems: A Case Study for Fluid/Structure Coupled Simulation', 12th ACM International Conference on Supercomputing, Melbourne, July 13-17 (1998).
3. Thomas Bönisch and Roland Rühle, 'Adapting a CFD code for metacomputing', 10th International Conference on Parallel CFD, Hsinchu/Taiwan, May 11-14, (1998).
4. W. Gropp, E. Lusk, N. Doss, A. Skjellum, 'A high-performance, portable implementation of the MPI message-passing interface standard', *Parallel Computing*, 22 (1996).
5. Graham E. Fagg, Jack J. Dongarra and Al Geist, 'Heterogeneous MPI Application Interoperation and Process management under PVMPI', in: Marian Bubak, Jack Dongarra, Jerzy Wasniewski (Eds.), 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', 91-98, Springer (1997).
6. Matthias Brune, Jörn Gehring and Alexander Reinefeld, 'Heterogeneous Message Passing and a Link to Resource Management', *Journal of Supercomputing*, Vol. 11, 1-17 (1997).
7. Thomas Beisel, Edgar Gabriel, Michael Resch, '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', *Lecture Notes in Computer Science*, 75-83, Springer (1997).
8. Matthias Müller and Hans J. Herrmann, 'DSMC - a stochastic algorithm for granular matter', in: Hans J. Herrmann and J.-P. Hovi and Stefan Luding (Eds.) 'Physics of dry granular media', Kluwer Academic Publisher (1998).

## Acknowledgements

The authors gratefully acknowledge support from PSC, KFA Jülich and the High Performance Computing Center Stuttgart.

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