

# Implementing MPI with Optimized Algorithms for Metacomputing

Edgar Gabriel, Michael Resch and Roland Rühle  
High Performance Computing Center Stuttgart  
Allmandring 30  
D-70550 Stuttgart  
Germany  
{gabriel, resch}@hlrs.de

*Abstract*— This paper presents an implementation of the Message Passing Interface called PACX-MPI. The major goal of the library is to support heterogeneous metacomputing for MPI applications by clustering MPP's and PVP's. The key concept of the library is a daemon-concept. We will focus in this paper on two aspects of this library. First we will show the importance of the usage of optimized algorithms for the global operations in such a metacomputing environment. And second we want to discuss whether we introduce a bottleneck by using daemon-nodes for the external communication.

*Keywords*— MPI, Metacomputing, Global Operations

## I. WHY ANOTHER MPI - IMPLEMENTATION ?

In the last couple of years a large number of tools and libraries have been developed to enable the coupling of computational resources, which may be distributed all over the world [5], [6], [8], [9], [12],[13]. The goal of such projects is usually to solve problems on a cluster of machines which cannot be solved by using a single Massively Parallel Processing System (MPP) or Parallel Vector Processors (PVP).

PACX-MPI (PARallel Computer eXtension)[10] is an implementation of MPI which tries to meet the demand of distributed computing. While most vendor implemented libraries do not support interoperability between different MPI-libraries, PACX-MPI makes the MPI-calls available across different platforms. The Interoperable MPI approach (IMPI) [11] may solve this problem, but still this is no standard and thus there are not yet any MPI-implementations according to these specifications. A lot of features as described in the IMPI draft document are however reflected in the PACX-MPI concept although it is also far from providing the full required functionality.

MPICH [7] as the mainly used MPI distribution supports a lot of platforms and supports the coupling of machines, too. The major disadvantage of this implementation is, that one may run in difficulties by coupling e.g. two machines with 512 nodes each, because of the number of open ports. In the worst case one may end up with 511 open ports on each node. The number of open ports is furthermore for all machines of importance, which are protected by some kind of firewalls. The less ports one has to use for the coupling of different resources, the easier is it to open and to control those few ports.

Some other approaches to achieve interoperability have been made. PVMPI [5] makes MPI applications run on a cluster of machines by using PVM for the communication between the different machines. Unfortunately the user can use only point-to-point operations and he has to add some non MPI congruent calls. The subsequent project, MPLConnect uses the same ideas but replaced PVM by a library called SNIPE [6], and supports now global operations too, in contrary to PVMPI.

A similar approach has been done by PLUS [2]. This library additionally supports communication between different message-passing libraries, like e.g. PARMACS, PVM and MPI. But again the user has to add some calls to his application. Another project called Stampi [12] has been recently presented. This project already uses the MPI2 process model, but focuses mainly on local area computing.

Referring to the experiences of a lot of those efforts, this paper presents the concept and results that were achieved with PACX-MPI especially with respect to optimized communication algorithms. The concept of the paper is as follows. The second section describes the main concepts and the main ideas of PACX-MPI. In the third section we focus on optimizing global operations for metacomputing. Afterwards we discuss in the fourth section whether we introduce a bottleneck by using daemon nodes for the external communication. In the fifth section we present some applications which used PACX-MPI during the Supercomputing'98 event in Orlando. Some optimization efforts are described there. In the last section we briefly describe the ongoing work and the future activities in this project.

## II. CONCEPT OF PACX-MPI

Before we start to describe the concept of PACX-MPI, we have to define, for what kind of clusters we want to use it. With PACX-MPI we do not intend to cluster workstations or even small MPPs or PVPs to simulate a big parallel machine. Our goal is to couple big resources to simulate machines, which can be hardly build nowadays. This includes that these machines are usually not in the same computing center and therefore we have to deal with latencies between the machines, which are in a complete different range than the latencies inside a single machine.

To couple different MPP's and PVP's, PACX-MPI has

to distinguish between internal and external operations. Internal operations are executed by using the vendor-implemented MPI-library, since these are highly optimized. Furthermore this is nowadays the only protocol, which is accessible on each machine and which can exploit the full capabilities of the underlying network of an MPP. Therefore PACX-MPI can be described as an implementation of MPI on the top of the native MPI-libraries.

External operations, e.g. point-to-point operations between two nodes on different machines, are handled by a different standard protocol. Actually PACX-MPI supports only TCP/IP, but we will add some other protocols like native ATM in the frame of a European project in the future. In this sense PACX-MPI can be described as a tool to provide multi-protocol MPI for metacomputing.

To avoid that each node has to open ports if it wants to perform some external operations, PACX-MPI uses two specialized daemon nodes for the external communication on each machine. Using these daemon nodes we can minimize the number of open ports and we can use fixed port-numbers. These two nodes are transparent for the application, and are therefore not part of global communicators, like e.g. `MPLCOMM_WORLD`. Figure 1 shows a configuration of two machines, each using 4 nodes for the application and how `MPLCOMM_WORLD` looks like in this example. On the left machine, which shall be the machine with the number one, the first two node with ranks 0 and 1 are not part of `MPLCOMM_WORLD`, since these are the daemon nodes. The next node with the rank 2 is therefore the first node in our global communicator and gets the global rank number 0. All other application nodes get a global number according to their local ranks minus two, the last node on this machine has the rank 3. On the next machine, the daemon nodes again are not considered in the global `MPLCOMM_WORLD`. The node with the local rank 3 is number 4 in the global communicator, since the numbering on this machine starts with the last global rank on the previous machine plus one.

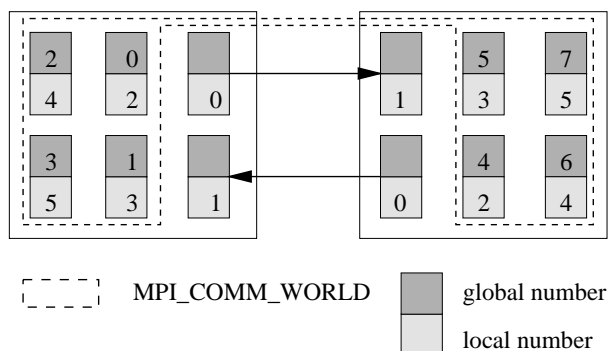


Fig. 1. An example of a two machine configuration coupled by PACX-MPI

To enable multi-protocol communication, PACX-MPI has to redirect MPI calls to PACX-MPI calls. These distinguish between internal and external communication. For

applications written in C, preprocessor directives redirect the MPI-calls to the library calls of PACX-MPI. For applications written in Fortran, the user has to ensure, that the Fortran-Interface of PACX-MPI is linked to his application before linking the native MPI-library, and thus his application will use this routines instead of the native MPI-calls.

For the startup of the metacomputer the user has to create a hostfile, which contains all used machines. Additionally the hostfile contains for each machine some information, especially the number of application nodes on each machine, the used protocol for the communication to this machine and optionally a startup command. For example a hostfile for coupling three machines called `host1`, `host2` and `host3` by using the TCP/IP protocol and using 100 nodes on each machine, the hostfile may look like this:

```
host1 tcp 100
host2 tcp 100 (rsh host2 mpirun -np 102 {exe})
host3 tcp 100 (rsh host3 mpirun -np 102 {exe})
```

Making use of the automatic startup facility of PACX-MPI one has to start only the application on the first machine, and the application will be started by the command given in the hostfile (usually based on `rsh` - operations) on all other machines. But one has also the possibility to start the application on each machine manually by simply replacing the startup command in the hostfile by the comment-sign (`#`).

The concept of PACX-MPI includes also data-compression for the external communication, data-conversion to support heterogeneous clusters and a buffering system to catch race conditions between different machines. These points together with the full concept of PACX-MPI are discussed extensively in [10].

#### A. Point-to-point operation in PACX-MPI

A point-to-point operation in this concept can be described as can be seen in Figure 2 the sending node has to check, whether the receiver is on the same machine. If it is on the same MPP, it can execute the `MPI_Send` command directly using the native MPI-library. If it determines, that the receiving node is on another machine, it sends the message to a daemon node. The daemon node transfers the message to the destination machine, where another daemon node receives the message and hands it out to the destination node. This way, all communication is bundled across only one connection link for each direction. Advantages and the problems of such a bottleneck are discussed later on.

### III. ANALYSIS OF GLOBAL OPERATIONS IN METACOMPUTING

#### A. Description of the problem

To achieve performance in a metacomputing scenario, where different machines are separated by tens of milliseconds of latency, it is very important to optimize every operation, especially global operations. Many algorithms,

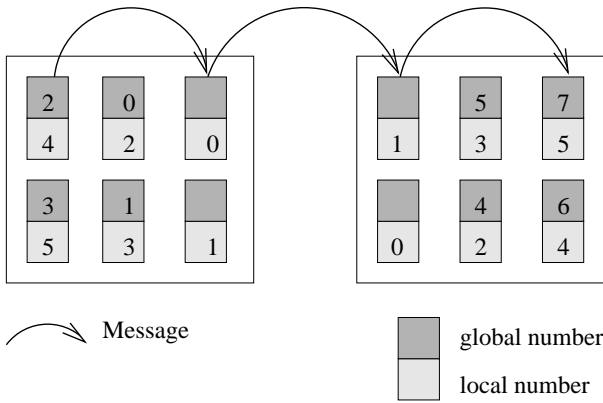


Fig. 2. Example for a point-to-point communication from global node 2 to global node 7

which seem to be optimal while executed on a single MPP, are not optimal when executed on a cluster of MPPs, because they are not optimized for different latencies. For example binomial tree algorithms are usually regarded to be optimal for broadcast-operations. In a metacomputing environment an algorithm, which does not minimize the number of external communication steps cannot be regarded as optimal, because its execution time is strongly dependent on the distribution of the nodes on the different machines. Figure 3 shows an example, where for a binomial-tree broadcast algorithm either one or two messages have to be sent between the machines, depending on how the nodes are distributed on both machines. To clarify

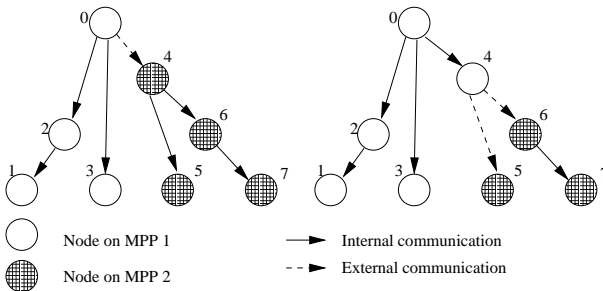


Fig. 3. Binomial-tree broadcast operation on eight nodes

how huge differences between communication steps for an internal and an external communication is, Table I shows some latencies for internal communication on different platforms and some measured latencies in metacomputing scenarios. Some more information on these tests can be found in [15].

The latency between two nodes on an MPP or PVP is usually in the range of a few microseconds. Even a somewhat older platform like the Intel Paragon has a latency of 41 microseconds. The latency between two Cray T3E, which are not situated in the same Computing Center, can be on the other hand in the range of tens to hundreds of milliseconds. In the worst case described above, the latency between the Stuttgart T3E and the Pittsburgh T3E is about 20000 times higher when using the standard in-

TABLE I  
COMPARISON OF LATENCIES FOR INTERNAL AND EXTERNAL COMMUNICATION

Connection	Network status	Latency [ms]
Cray T3E internal	internal	0.016
NEC SX4 internal	internal	0.009
IBM SP2	internal	0.038
Intel Paragon	internal	0.041
Stuttgart T3E - Pittsburgh T3E	dedicated	70 - 75
Stuttgart T3E - Pittsburgh T3E	internet	~ 330
Stuttgart T3E - Juelich T3E	internet	~ 29
Stuttgart T3E - Manchester T3E	internet	~ 48

ternet path compared to the internal latency of the Cray T3E.

### B. The PACX-MPI broadcast and reduce algorithm

The broadcast algorithm of PACX-MPI can be briefly described as shown in Figure 4. The root-node sends a message to all other machines, which have nodes participating in the communicator used. Then it performs a broadcast operation to the local part of the communicator. On all other machines, PACX-MPI defines a local root node, and this root node will execute the broadcast operation on its machine. This algorithm, still not optimal, makes sure that exactly one message will be sent to each machine, and is by the way very similar to the algorithm proposed by the IMPI Steering Committee [11].

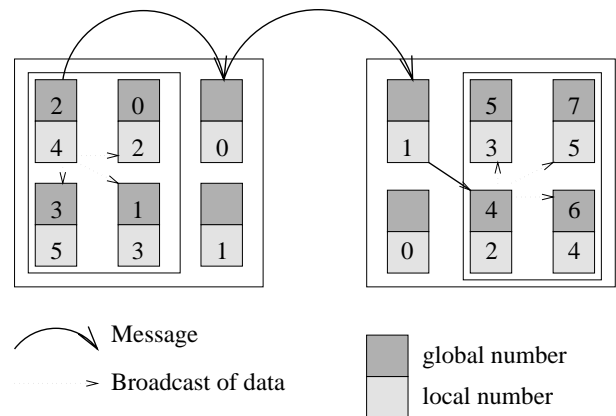


Fig. 4. Broadcast operation of PACX-MPI

The reduce algorithm of PACX-MPI is very similar. In a first step, a reduce operation on the local part of each communicator is executed, defining again on each machine a root-node. Each local root sends now the local result to the global root and the global root, specified by the application, calculates the global result.

### C. Comparison of the PACX-MPI algorithms to tree-based algorithms

In the following we give comparisons of different algorithms to show the relative performance for PACX-MPI. The tests were made on a single Cray T3E but simulating two machines. Two completely independent MPI-executables were started on the same machine, and the communication between those two programs was done by using the TCP/IP Interface of the T3E. Thus we simulated a metacomputing environment on a single machine. The latency for an external communication can be usually split in two parts:

$$latency = t_{I/O} + t_{trans} \quad (1)$$

with  $t_{I/O}$  being the time spent by the network-interface node to locally set up a message and  $t_{trans}$  being the time to physically transfer a message across the link. For coupling two partitions on the same machine,

$$t_{trans} = 0;$$

and

$$latency = t_{I/O}. \quad (2)$$

For a Cray T3E  $t_{I/O} = 3.5ms$  to send a complete PACX-MPI message from one partition to the other, considering that we are transferring two messages, a header and the data.

In Table II we compare the execution times for broadcast and reduce operations on 64 nodes with a message size of 1KByte for different methods.

TABLE II  
COMPARISON OF THE EXECUTION TIMES OF DIFFERENT BROADCAST AND REDUCE ALGORITHMS DEPENDING ON THE DISTRIBUTION OF NODES ON TWO MACHINES.

MPP 1	MPP 2	Ext. comm.	P-B [ms]	T-B [ms]	P-R [ms]	T-R [ms]
32	32	1	3.74	3.91	9.6	9.7
16	48	2	3.73	7.60	10.7	14.4
8	56	3	3.66	11.6	10.7	18.6
4	60	4	3.70	14.7	10.2	23.2
2	62	5	3.71	18.5	9.76	27.4
1	63	6	3.72	21.5	10.3	31.2

The first (MPP 1) and the second (MPP 2) column in this table give the number of nodes used on each MPP. The third column (Ext.comm) indicates, how many external communication steps have to be performed in a tree algorithm. The fourth column gives the execution time for a PACX-MPI-broadcast operation(P-B), the fifth column the time for a binomial-tree broadcast (T-B) algorithm. The sixth and seventh column represent the execution time for PACX-MPI-reduce(P-R) and tree-based Reduce (T-R) operation respectively on the same number of nodes. We measured the times on each node. Values given in the table are maximum values over all nodes measured. We did

a series of tests for each algorithm. Minimum values for those series are given in the table.

The first result that can be seen in Table II is that the metacomputing latency is the dominating part of the operation. Times for local broadcasts or reduces typically increase with increasing message size but remain in the range of microseconds. So message size does not significantly change the results seen here. The main result of Table II is, that a broadcast operation of PACX-MPI is nearly constant independent of the number of nodes on each machine, since the algorithm makes sure, that for every configuration exactly one message is sent between both machines. Time for a binomial-tree broadcast algorithm can be calculated approximately as the number of external communication steps  $n_{ext.comm}$  times the latency between the machines, considering equation (2):

$$t_{T-B} \approx n_{ext.comm} \cdot latency \quad (3)$$

The results show, that this is also what we measure. The latency between both machines dominates both algorithms, and therefore an optimal algorithm for such an environment has to minimize the usage of this connection. The table above could give the impression, that the smaller the number of nodes on a machine, the more external communication steps have to be performed. But this impression is wrong, because e.g. with 16 nodes on the first machine, 2 messages have to be sent for a tree-based algorithm, but with 17 nodes on the first machine already 5 messages have to be sent between both machines.

The results are similar for the reduce operation. The reduce operation additionally requires in both algorithms an overhead of about 6 ms on these machines, therefore the execution time for the PACX-MPI-reduce operation can be calculated approximately as

$$t_{P-R} \approx 6ms + latency \quad (4)$$

and for a tree reduce operation as

$$t_{T-R} \approx 6ms + n_{ext.comm} \cdot latency. \quad (5)$$

The overhead of about 6ms for the reduce operation points out another problem of such measurements. To avoid, that the startup of the metacomputer has any effect on the measured times, we have to perform a barrier operation before all tests made.

The barrier operation in PACX-MPI can be explained as shown in Figure 5. First each machine is executing a local barrier on the local part of the communicator using the MPI\_Barrier routine of the native MPI-library. Then one node on each machine contacts the smallest node in this communicator. When this node has received a message from each machine, which have nodes participating in this communicator, it again sends a message back to each machine. All other nodes on each machine are waiting in a second barrier until the dedicated node receives the message from the node with rank 0 in this communicator.

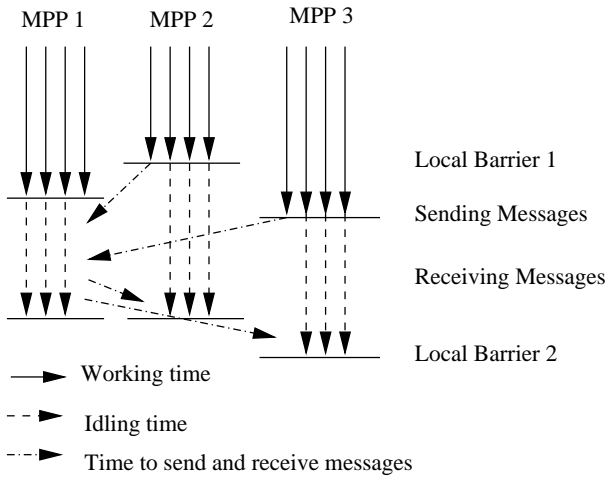


Fig. 5. Barrier algorithm of PACX-MPI

The barrier operation in PACX-MPI makes sure, that each process can continue its work only after all processes entered this routine, but it cannot synchronize perfectly. Because of this problem the nodes on the first machine can continue their work about one external communication step earlier than the nodes on the second machine (because a message has to be sent to all other involved machines). This is no problem for the broadcast algorithm. After having passed the barrier the root node may issue the broadcast call. The receiving nodes may start their work later but this does not effect the timing results. For the reduce operation, the root node has to wait for the result of the other machines. Since nodes on the other machine have to wait in the barrier much longer than the root node, the root node will measure some overhead waiting time before senders can actually start to work. This overhead time should be exactly the time that it takes for the confirming message of the barrier to be sent to all other machines. In fact, if we examine where the algorithm spends its time, then we can see, that most of the time is spent in the receive-operation for the message from the other machine.

### C.1 Wide area networks

Table III presents some tests for the reduce-operations performed between the Stuttgart Cray T3E and the Cray T3E of the Pittsburgh Supercomputing Center. For these tests a dedicated ATM-link between both T3E's was used with a bandwidth of 10 MBit/s. Latency now has to be split into  $t_{I/O}$  which still is around 3.5 ms and  $t_{trans}$  which depends on the link and is around 70ms. The result for the PACX-MPI-reduce operation is now again constant and independent of the configuration of the metacomputer. But the results of the tree-reduce algorithm can not be described by equation (5). The results for the tree-based reduce-operation can be described by following equation:

$$t_{T-R} = 2 * t_{trans} + 6ms + (n_{ext.comm}) * t_{I/O} \quad (6)$$

with  $2 * t_{trans}$  being the time to deliver one message from Stuttgart to Pittsburgh, considering that we have to deliver two messages. One for the header and one for the

TABLE III  
REDUCE TESTS BETWEEN STUTTGART AND PITTSBURGH

MPP 1	MPP 2	Ext. comm.	P-R [ms]	T-R [ms]
32	32	1	146.8	148.5
16	48	2	146.0	151.8
8	56	3	145.8	155.5
4	60	4	145.8	160.5
2	62	5	146.0	163.3
1	63	6	146.3	168.0

data. This will certainly optimized in the future by using a short protocol for short messages. This equation holds only as long as we do not use the complete bandwidth of the dedicated link by a single message. Only in that case messages can be sent in a pipelined mode hiding away part of the latency.

### D. Coupling more than two machines

Another test to prove, how important it is to optimize global operations, is to show how the execution times of broadcast-operations change with a constant number of nodes but varying number of machines. For such an operation the execution time for a PACX-MPI-broadcast operation ( $t_{P-B}$ ) can be approximated by

$$t_{P-B} \approx \sum_{i=1}^N latency_i \quad (7)$$

and time for a binomial tree broadcast-algorithm ( $t_{T-B}$ )

$$t_{T-B} \approx \sum_{i=1}^N \sum_{j=1}^{n_{ext.comm.}} latency_i \quad (8)$$

with  $N$  being the number of connections and  $latency_i$  being the latency for connection on link  $i$ <sup>1</sup>. Since for the PACX-MPI-broadcast algorithm the number of external communication steps is always 1, it is very easy to prove, that theoretically

$$t_{P-B} \leq t_{T-B}. \quad (9)$$

These tests were executed again for 96 nodes by simulating two or more partitions on a single machine. Therefore the assumptions for the latency of equation (2) are still valid. The message size was again 1KByte. In Table IV the result for two machines is nearly identical between both algorithms. In both cases only one external communication has to be performed and therefore the results are similar. Distributing 96 nodes on three machines requires for the PACX-MPI-broadcast algorithm two external communication steps - one message to each machine. But it requires

<sup>1</sup>In both cases - depending on the protocol used and on the size of messages sent - it may happen that one does not see the full latency but only  $t_{I/O}$ . For the sake of simplicity, this case is not further investigated here. Future investigations will have to deal with this problem.

TABLE IV  
BROADCAST OPERATION WITH 96 NODES AND VARYING NUMBER OF  
MACHINES

Number of machines	P-B [ms]	T-B [ms]
2	3.77	4.11
3	4.58	25.4
4	8.68	8.04

seven external messages for the binomial-tree broadcast algorithm. The execution time for PACX-MPI-broadcast algorithm is somehow smaller than expected, but this behavior can be explained with some pipelining effect for the external communication in this case. For four machines, the result for both algorithms are again similar, because again both algorithms have to perform three external communication steps.

#### E. Further optimization possibilities

One may try to further optimize the broadcast algorithm of PACX-MPI by using a tree-algorithm to distribute the data between the machines, and executing a broadcast operation on each machine afterwards. This algorithm is presented in Figure 6. The algorithm for the local broadcast is not explained in this picture, because usually it is really unimportant compared with the algorithm to distribute the data between the machines. Usually the algorithm shown in Figure 6 will not be faster than the PACX-MPI-broadcast algorithm. PACX-MPI sends a message to all other machines. Time for such a communication is

$$t_{I/O} * (N - 1) + 2 * t_{trans} \quad (10)$$

while an algorithm as described would need

$$O(\log(N - 1) * 2 * (t_{I/O} + t_{trans})) \quad (11)$$

where  $N$  now is the number of machines used. For large  $t_{trans}$  - which typically is the case in wide area networking - PACX-MPI will in most situations be faster. But nevertheless there may be some situations, where the algorithm in Figure 6 may be faster than the algorithm of PACX-MPI, especially when we have to deal with a mixture of dedicated network connections and standard internet paths.

Up to three machines, the algorithm of Figure III-E and the algorithm of PACX-MPI are identical.

Another optimization possibility for wide area metacomputing would be to distinguish between the latencies of the different machines. By coupling for example some machines in Europe to a single machine in the US, it would make sense to minimize the communication between the US machine and the machines in Europe. Therefore a broadcast operation, which has the root-node on the US machine should send only a single message to Europe, and another machine in Europe should distribute the data to all other

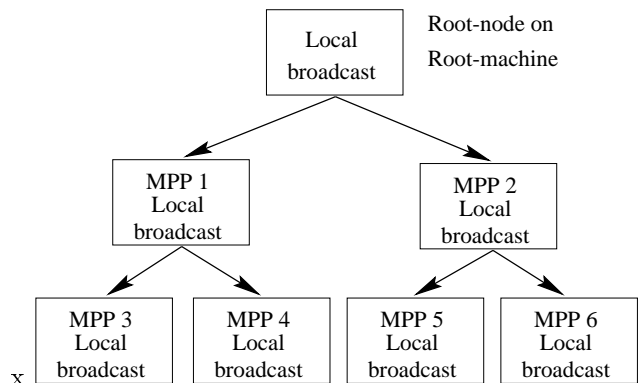


Fig. 6. Optimized Broadcast-algorithm in a metacomputing scenario

machines. This algorithm would require a kind of dynamic resource-management, which is not integrated in PACX-MPI at the moment.

#### IV. IS THERE A BOTTLENECK BECAUSE OF THE USAGE OF DAEMON NODES ?

Another important question with respect to the concept of PACX-MPI is, whether the library introduces a bottleneck by using two daemon nodes for the communication between nodes on different machines. This introduces a bottleneck through which all communication traffic has to go. Especially for communication intensive applications one might expect to see problems of overflow or contention. First of all we have to remark, that for most situations different machines are coupled by a single physical network connection, thus every message has to use this connection no matter of whether they are communicating directly or going through a daemon node first. With respect to the latency the physical latency of the network between the machines is dominant, and therefore the two steps of internal communication, from the sender node to a daemon node and on the destination machine from the communication node to the receiving node, are not relevant.

Coupling more than two machines one may use different physical links between different machines. But even this is often only possible when coupling local resources. Additionally lot of machines provide only one networking interface node, and therefore the whole communication has to pass a single node, independent of whether a node tries to communicate directly to another node or by using daemon nodes. Especially in those situations where there is only a single interface node available on a machine, it seems to be useful to be able to handle the bottleneck by a daemon rather than to let the system do it. In doing so the daemon can take care of communication problems. Furthermore, the daemon concept allows to quickly adapt to situations where there are more interface nodes available. In such a case one would introduce as many daemons as interface nodes in order to load balance communication.

In this chapter we want to examine, how the daemon nodes behave in a situation, where all nodes try to send messages at the same time to another node. Such situations often occur for example when doing online-visualization or

when executing postprocessing. In this case one node collects data from all other processes. In a metacomputing scenario this would mean that all nodes of one machine try to send messages to one daemon to transfer data to a second machine. There is obviously the risk to overload the daemon in such a situation.

In a first test we are comparing the execution times on the receiving node for both situations. Table V shows the results of these tests with varying number of nodes on the second MPP. This test was performed again simulating two partitions on the same Cray T3E in Stuttgart. The size of the data was again 1KByte, all communication steps had to pass the external link.

TABLE V  
COMPARISON OF DAEMON AND DIRECT COMMUNICATION

Nodes on MPP 2	time for daemon communication [ms]	time for direct communication [ms]
16	73.0	56.2
20	90.3	70.2
24	107.1	82.5
28	123.9	95.3

The first column gives the number of nodes on the second MPP, the first machine had always only one node to receive all messages. The second and third column indicate the total execution time on the receiving node for receiving all messages with PACX-MPI respectively direct socket-communication. The execution times with PACX-MPI are in all cases higher than for direct communication.

Lets examine one of these cases more detailed. When 28 nodes try to communicate with node 0, the receiver needs about 4.425 ms to receive a message in average with PACX-MPI and about 3.4 ms with direct socket-communication. Thus PACX-MPI introduces an overhead of 1 ms per message.

Examining every single communication step we have a very similar behaviour in both cases. To receive the first message it takes between 12.7-23.0 ms. Most of this time can be explained again with the asynchrony of both machines due to a barrier operation. All following messages need about 3.7-5.0 ms for PACX-MPI and about 2.6-3.7 ms for the direct socket-communication.

On the sender side things are slightly different. Since PACX-MPI works in the background with incomplete buffered sends, execution times for the sender with PACX-MPI are much smaller than for direct socket communication. Additionally the sender in PACX-MPI is only dealing with internal communication, since it sends the message to the daemon nodes. Therefore the sending nodes in PACX-MPI are dealing with times between 0.5 -2.0 ms, depending on how fast they get access to the daemon node, compared to up to 10 ms for direct socket communication, depending on how fast they get access to the network I/O node.

To resume this chapter, for many-to-one situations with PACX-MPI we introduce an overhead of about 1 ms per

message on the receiving side, and we are up to ten times faster on the sending node.

## V. APPLICATIONS AND RESULTS

During the Supercomputing 98 event in Orlando, several applications have been presented using PACX-MPI. For these tests the Cray T3E of the Pittsburgh Supercomputing Center and the Cray T3E of the High Performance Computing Center Stuttgart have been coupled using a dedicated transatlantic link with a bandwidth of 10 MBit/s and a latency of about 75 ms.

### A. URANUS

The first application is a CFD-code called URANUS (Upwind Relaxation Algorithm for Nonequilibrium Flows of the University of Stuttgart) [3]. This program has been developed for simulating the reentry of a space vehicle in a wide altitude velocity range. The reason why URANUS was tested in such an environment is that soon two additional components of URANUS will have a great demand on memory: the nonequilibrium part has been finished in the sequential code and will be parallelized soon. Additionally we will simulate the Crew-Rescue-Vehicle (X-38) of the new international space-station with more than 3 Million cells. Both components together require memory in the range of hundreds of Gigabytes, that cannot be provided by a single machine today. During the SC98 we simulated the European space-vehicle HERMES with 1.7 Million cells using 992 CPU's on two Cray T3E's.

URANUS consists mainly of three parts. The first part (preprocessing) reads the input data and distributes all information to the application nodes. This part has not been parallelized up to now, because of the lack of MPI based parallel I/O on some platforms. It has been adapted, such that two nodes are now distributing the data. This avoids to transfer huge amounts of data over the transatlantic link.

The second part is the processing part. To optimize latency-hiding we do some kind of message-pipelining. In each iteration the receiving nodes check whether a message has arrived. If so the message is received and the iteration is continued using the newest results. If the new results have not arrived yet due to the large latency then the process just continues the iteration with the older values it got from the last iteration. The results presented in [3] show, that using this concept, the time for an iteration is similar to the original iteration time for URANUS on a single MPP, but the number of iterations for convergence has to be increased by about 10 percent.

The third part finally does postprocessing including some calculations and the writing of the results on disk. This part has not been parallelized yet. However, we plan to couple the code to a distributed visualization tool. This would allow to process data for visualization in parallel. The data transfer would be substantially reduced by sending only that information which is relevant for the visualization, like pixel information.

### B. P3T-DSMC

The second application is called P3T-DSMC ( Parallel/Physics/Particle 3D Tools - Direct Simulation Monte Carlo). Based on the object-oriented P3T toolkit, developed by the Institute for Computer Applications of the University of Stuttgart, this code is used for general particle tracking problems. This Monte Carlo Code performs excellent on a metacomputer, since the computation to communication ratio is very small.

For a small number of particles per CPU - in the range of several thousand - the code shows some overhead for metacomputing due to the much higher latency between the machines, compared with the internal latency of a MPP. But for more than 15000 particles and up to about 500000 particles per CPU, the applications shows no overhead. The execution times are identical to those on a single machine.

More results for both URANUS and P3T-DSMC can be found in [4]

### C. P3T-MD

The third application is also based on the P3T-toolkit, but instead of a Monte-Carlo code this program solves the molecular-dynamic equations to simulate the interactions between the particles. Therefore the code is stronger coupled compared to P3T-DSMC. During the SC98 event, a lot of tests have been performed with both P3T applications using up to 1024 processors. The results could not yet be fully evaluated.

### D. Some results of the network throughput

During the Supercomputing 98 a tool called linkTV enabled to watch the traffic on the dedicated link. Therefore the behavior and the performance of different applications could be observed. Figure 7 shows a diagram where the communication performance of URANUS, P3T-DSMC and P3T-MD is shown in measured bandwidth [KB/s]. The figure shows sustained bandwidth for both directions. In many cases the communication pattern is symmetric. To get an understanding of the behavior of the link one has to add both lines shown in the figure.

While URANUS uses around 900 Kbytes/s peak, considering that in both directions the bandwidth is about 450 KB/s, P3T-DSMC uses less of the bandwidth - in the range of 150 KB/s. Some tests have shown, that this may occur due to the fact, that P3T-DSMC used the data-compression for the external communication while URANUS did not. Using data compression the amount of data to be sent can be reduced by a factor of 3 typically, which for these simulations resulted in less traffic on the link and thus lower measured bandwidth. From a networking point of view this may look bad since the full capacity of the link is not used. However, the global performance of the code can be improved because overall data transfer times are reduced. This, in consequence, reduces overall simulation time.

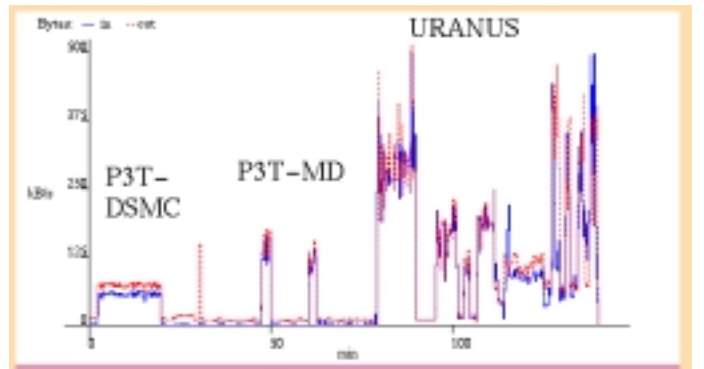


Fig. 7. Snapshot from the dedicated Network link between Stuttgart and Pittsburgh

## VI. OUTLOOK

The ongoing work of PACX-MPI is strongly coupled to two different projects. First PACX-MPI will be part of a project called METODIS (METacomputing TOols for DIstributed Systems), which is supported by the European Community. The goal of this project is to develop a set of tools for metacomputing, including a metacomputing MPI, a general ATM interface that will be used by PACX-MPI and a metacomputing version of the performance analysis tool VAMPIR that will be coupled to PACX-MPI. Therefore the complete MPI-1.2 standard and some parts of MPI-2 will be implemented during the next time for this project.

Like also mentioned above a multiprotocol interface for PACX-MPI will be developed to support not only TCP/IP but also some other network protocols, especially ATM and HIPPI. Therefore a cluster may use an ATM-connection between 2 different machines if available, and TCP/IP for all other machines. To support this, some kind of connection-based resource management will be added.

Some of the measurements above showed also still some performance problems and some improvement potentials for global operations. For example some more work has to be done for the barrier operation and the introduction of a short-message protocol for TCP/IP would make sense.

The work of the last two years pointed out, that applications have to be adapted for metacomputing, since dealing with latencies of different ranges require very latency tolerant algorithms. Although a lot of work has been done already in this field, this may be another major issue to focus on.

The second project that strongly influences the further development of PACX-MPI will be the IMPI standardization. As already explained many concepts described in this standard are already reflected in PACX-MPI concepts - like the daemon concept or the distributed handling of global operations. Furthermore a project like PACX-MPI can only profit from a standardization of issues that concern coupling of heterogeneous architectures. Nevertheless there are a lot of questions that have to be addressed. First of all it is unclear when IMPI will be adopted and implemented by hardware vendors. Therefore an idea would be to change and use PACX-MPI in a way, that it can serve as

an interoperable MPI until the vendor-implemented MPI fulfills the IMPI criteria. Second the lessons learned from PACX-MPI might be helpful to further improve the IMPI standard.

## VII. SUMMARY

We've presented in this paper an implementation of the Message Passing Interface for a very sophisticated meta-computing - environment. Comparing some results for broadcast and reduce operations we have proved the importance of minimizing the number of external communications steps to achieve performance for such a scenario. We have also shown that by using two daemon nodes for the external communication we can improve the performance of the external send-operations and we only add a small overhead per operation at the receiving side. During the last 2 years and especially during SC'98 in Orlando a couple of applications have shown that using this library one can indeed make his application run on a metacomputer.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the help of all members of the PACX-MPI group. Furthermore we acknowledge the helpful support by networking organizations and groups, especially STAR TAP. In addition we would like to thank Pittsburgh Supercomputing Center and the High Performance Computing Center Stuttgart for providing their machines for our tests and demonstrations.

## REFERENCES

- [1] Massimo Bernaschi, Giulio Iannello, *Collective Communication Operations: Experimental Results vs. Theory*, Concurrency: Practice and Experience, 10(5):359-386, April 1998. Online version available at <http://www.grid.grid.unina.it/iannello/dapaa.htm>
- [2] Matthias Brune, Jörn Gehring and Alexander Reinefeld, *Heterogeneous Message Passing and a Link to Resource Management*, Journal of Supercomputing, Vol. 1, 1-17 (1997).
- [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] Th. Eickermann, J. Heinrichs, M. Resch, R. Stoy, R. Völpel, *Metacomputing in gigabit environments: Networks, tools and applications*, Parallel Computing 24 (1998) 1847-1872.
- [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] Graham E. Fagg, Keith Moore, Jack J. Dongarra, Al Geist, *Scalable Networked Information Processing Environment (SNIPE)*, Technical Paper, Supercomputing 1997.
- [7] Ian Foster, Jonathan Geisler, William Gropp, Nicholas Karonis, Ewing Lusk, George Thiruvathukal, Steven Tuecke, *Wide-Area Implementation of the Message Passing Standard*, Parallel Computing, 24 (1998).
- [8] Ian Foster, Carl Kesselman, *The Globus Project: A Status Report*, Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pg. 4-18, 1998.
- [9] Ian Foster, Jonathan Geisler, Steven Tuecke *MPI on the I-WAY: A Wide-Area, Multimethod Implementation of the Message Passing Interface* Proc. 1996 MPI Developers Conference, 10-17, 1996.
- [10] Edgar Gabriel, Michael Resch, Thomas Beisel, Rainer Keller, *Distributed Computing in a Heterogeneous Computing Environment*, in Vassil Alexandrov, Jack Dongarra (Eds.) 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', 180-188, Springer, 1998.
- [11] IMPI Steering Committee, *IMPI - Interoperable Message-Passing Interface*, information can be obtained from <http://impi.nist.gov/IMPI>, DRAFT processed on November 5, 1998.
- [12] 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.
- [13] Thilo Kielmann, Rutger F.H. Hofman, Henri E. Bal, Aske Plaat, Raoul A.F. Bhoedjang, *MagPie: MPI's Collective Communication Operations for Clustered Wide Area Systems*, submitted for publication in 1998, online version available at <http://www.cs.vu.nl/albatross>
- [14] 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.
- [15] Michael Resch, Holger Berger and Thomas Bönisch *A Comparison of MPI Performance on Different MPP's*, in Marian Bubak, Jack Dongarra, Jerzy Wasniewski (Eds.) 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', 25-32, Springer, 1997.