Performance Issues of Intercontinental Computing

Michael Resch, Thomas Beisel, Thomas Boenisch
University of Stuttgart
Allmandring 30
D-70550 Stuttgart
Germany
resch@rus.uni-stuttgart.de

Bruce Loftis, Raghu Reddy
Pittsburgh Supercomputing Center
4400 Fifth Avenue
Pittsburgh, PA 15213
412/268-8216
bruce@psc.edu, rreddy@psc.edu

ABSTRACT:

This paper presents a status report on our efforts and experiences with a metacomputing project that aims at connecting a Cray T3E at Pittsburgh Supercomputing Center and a Cray T3E at the High Performance Computing Center in Stuttgart. We describe how the two machines are configured into a single virtual machine running code with standard MPI calls. The library that handles the underlying communication management is presented. This library provides the user with a distributed MPI environment with most of the important functionality of standard MPI.

The first application we have run on the coupled T3E's is the flow simulation package URANUS. Problems and strategies with respect to metacomputing are discussed briefly for this code. First results both for the library performance and the code performance are given for tests on both local machines and machines connected via vBNS. These tests will serve as a starting point for further development. Investigation of these results shows that latency will be the critical factor for all applications except those that are embarrassingly parallel.

KEYWORDS:
Metacomputing, Distributed MPI, Flow simulation, Latency, vBNS, Cray T3E

Overview

A difficulty in simulating very large physical systems is, that even massively parallel processor systems (MPP) with a large number of nodes may have not enough memory and/or not enough performance. There are many examples of these *grand-challenge* problems: CFD with chemical reactions or crash simulations of automobiles with persons inside. We can use distributed computing or *metacomputing* to execute these very large models by combining two or more MPP's in one single cluster.

In summer 1996 a G7 meeting motivated an attempt to couple Cray T3E's at Pittsburgh Supercomputing

Center (PSC) and the University of Stuttgart (RUS) across the Atlantic Ocean to establish a virtual system with 1024 nodes and a theoretical peak performance of 675 GFlops. The PACX library [1] from RUS was used to allow an application to use standard MPI calls and work on this virtual machine. At Supercomputing'96 it was shown that the approach worked for a small demonstration experiment.

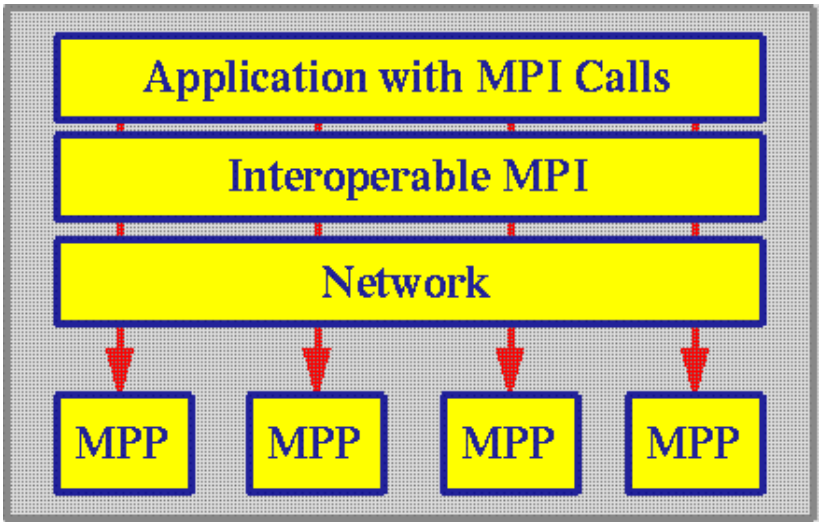
This was a starting point for further work. We are presenting our preliminary progress here, and during SC'97 we expect to demonstrate a real application running on an transatlantic virtual machine. The San Diego Supercomputing Center has also provided T3E time, and we intend to explore connecting 2 or 3 T3E's using the vBNS.

The application used is a flow solver developed at the University of Stuttgart [2] and adapted for parallel computation by RUS [3]. It will be coupled to a visualization tool developed by RUS to allow collaborative working and visualization [4]. These collaboration tools will further increase the communication needs of the whole scenario. Therefore it will become crucial for the underlying communication software to reduce latency as much as possible. Overlapping of communication and computation at the application level will be an even more crucial factor for this project.

Interprocessor Communication Methods

To reach the metacomputing goals, a library called PACX (PArallel Computer eXtension) was developed at RUS as an extension of the message passing library MPI. Initially PACX was developed to connect an MPP and a fileserver to allow fast data transfer from inside the application. This first version of PACX was based on raw HiPPI to exploit the underlying network. The next step was extending PACX to connect two MPP's. Driven by the idea of running one application on two MPP's the goal was no longer to just send or receive data from a fileserver but to have one MPI application exploit the full potential of two fast machines. The goal was to do this distributed computing without requiring the user to change the application code.

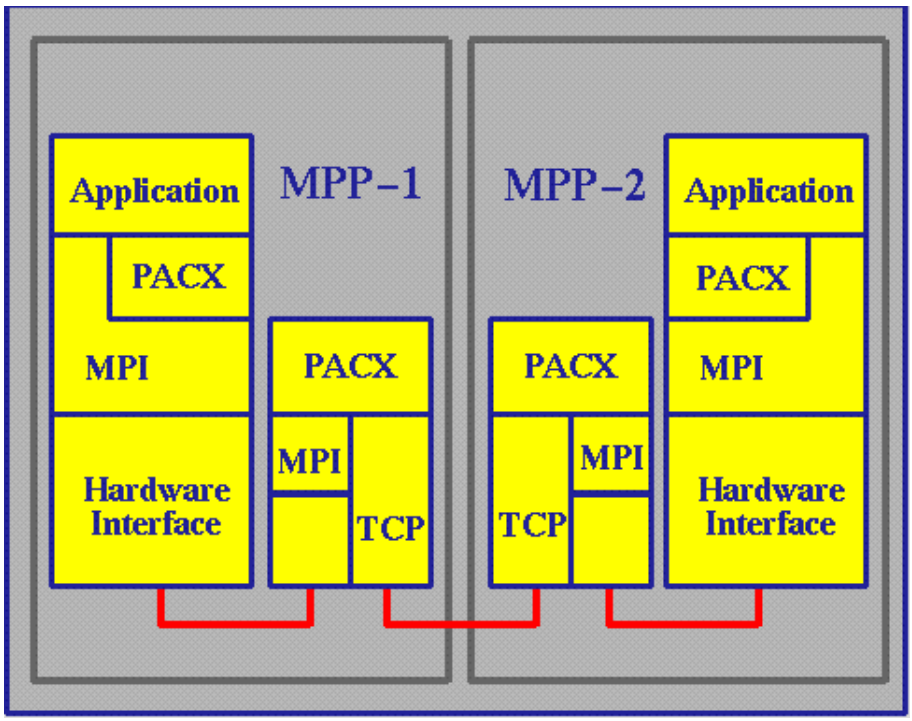
The design decision was to implement PACX as a library. It is implemented between the application and the local MPI implementation.



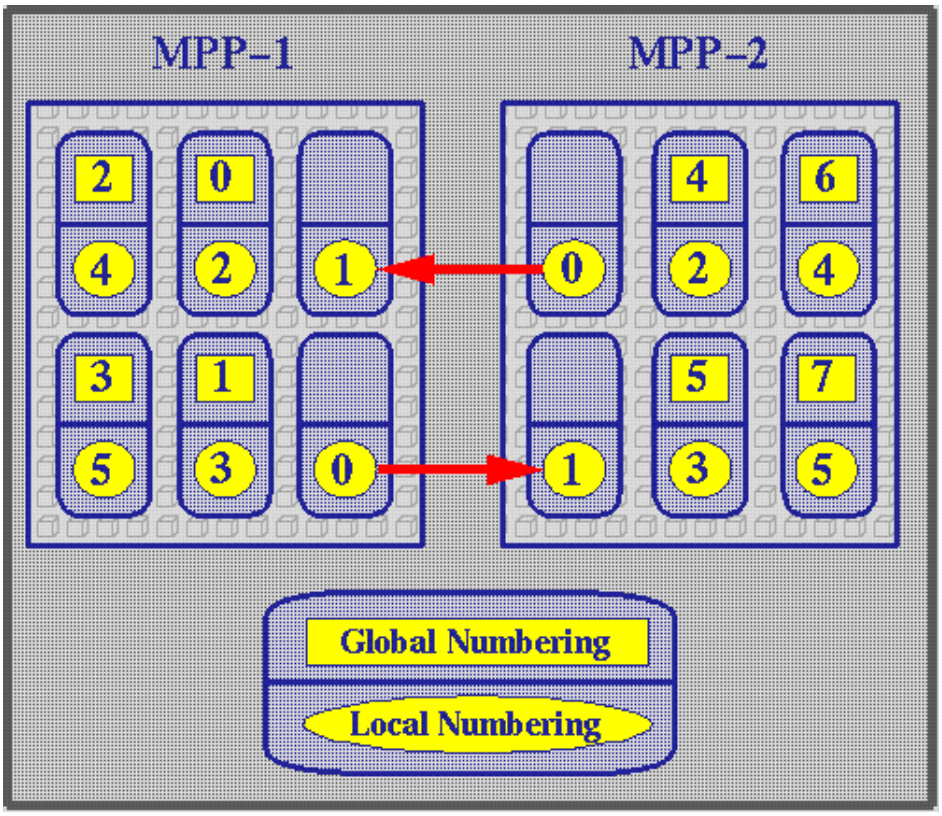
The main idea was that the application would not have to be changed to run on two or more MPP's. It

would just have to use normal MPI calls. Naturally, an efficient application has to be aware that it is running in a distributed computing environment. This imposes restrictions like low bandwidth and high latency that have to be considered by the algorithm. However, unlike other tools that are available (PVMPI,PLUS), the user does not have to insert any extra calls to the code. With PACX, MPI calls are intercepted and diverted into the PACX library. The PACX library determines whether there is a need for contacting the second MPP. If not, the library passes the MPI call to the local system unchanged, where it is handled internally. This guarantees usage of highly tuned vendor specific MPI implementations for internal communication. The overhead PACX imposes is very small on the CRAY T3E so that for calls on the local machine hardly any difference in performance can be seen.

For the communication between the two machines, each side has to provide two additional nodes, one for each communication direction. These nodes are transparent to the application. The responsibility of each communication node is to receive data from the compute partition of its machine or from the network, to compress or uncompress the data, and then transfer data to the network or the local compute nodes. Data compression uses the library ZLIB.



Communication via the network is done with TCP sockets. The concept of using two extra communication nodes (one for handling all outgoing communication and one for the incoming communication) has turned out to be a useful design as we plan to extend the concept to more than two machines. In addition it was easier this way to handle the complex communication needs of a real world application.

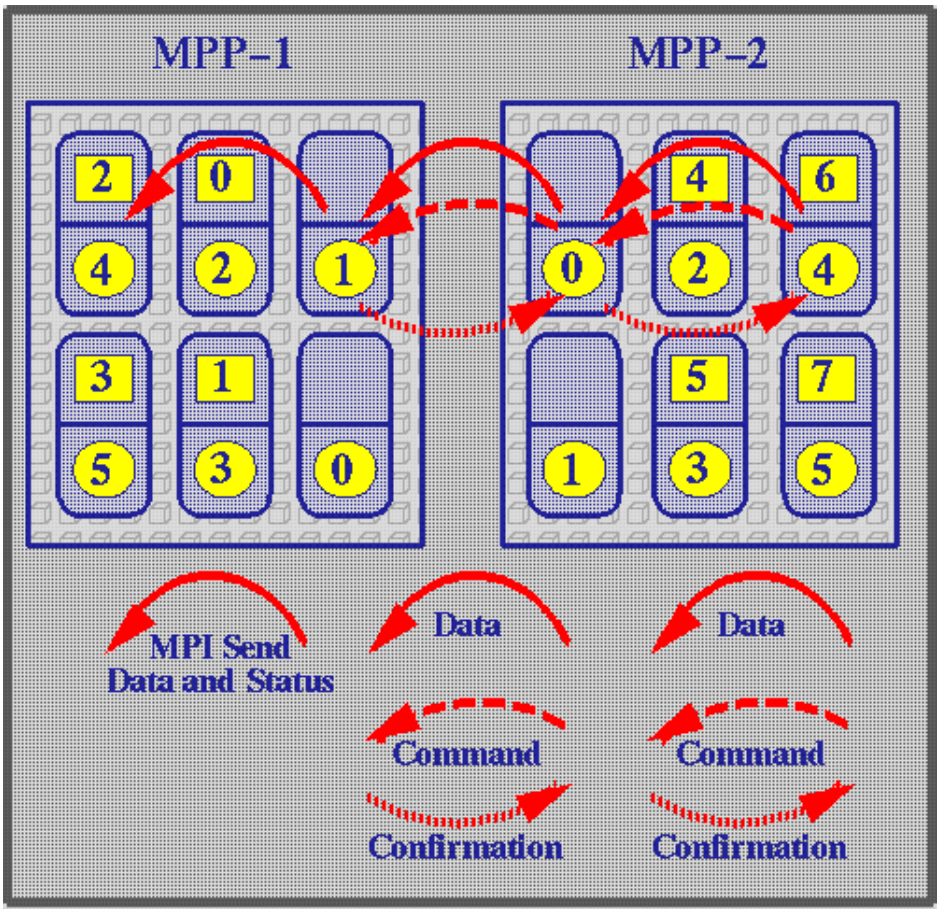


Like MPI, PACX has language bindings for FORTRAN 77 and ANSI C. But while MPI consists of more than 120 function calls PACX was restricted to a smaller number. It mainly implements those functions that are the most frequently used ones and omits the obscure ones. At this time PACX supports the following calls:

Initialization and control of the environment:
MPI_Init, MPI_Finalize, MPI_Abort, MPI_Comm_rank, MPI_Comm_size
Point to point communication:
MPI_Send, MPI_Bsend, MPI_Recv with all datatypes
Collective operations:
MPI_Barrier, MPI_Bcast with all datatypes,
MPI_Reduce and MPI_Allreduce with operations MPI_SUM,MPI_MAX,MPI_MIN for all FORTRAN 77 datatypes and for the ANSI C datatypes MPI_FLOAT, MPI_DOUBLE and MPI_INT
Nonblocking Communication:
MPI_Isend, MPI_Irecv

In addition to these calls, communicator constructs have been implemented and are currently in the testing phase. These will allow normal usage of communicator constructs across the machines without restrictions. An application may well have 7 nodes on one machine and 13 nodes on the second that together form a self-defined communicator.

The handling of a send/recv operation is shown in the next figure.

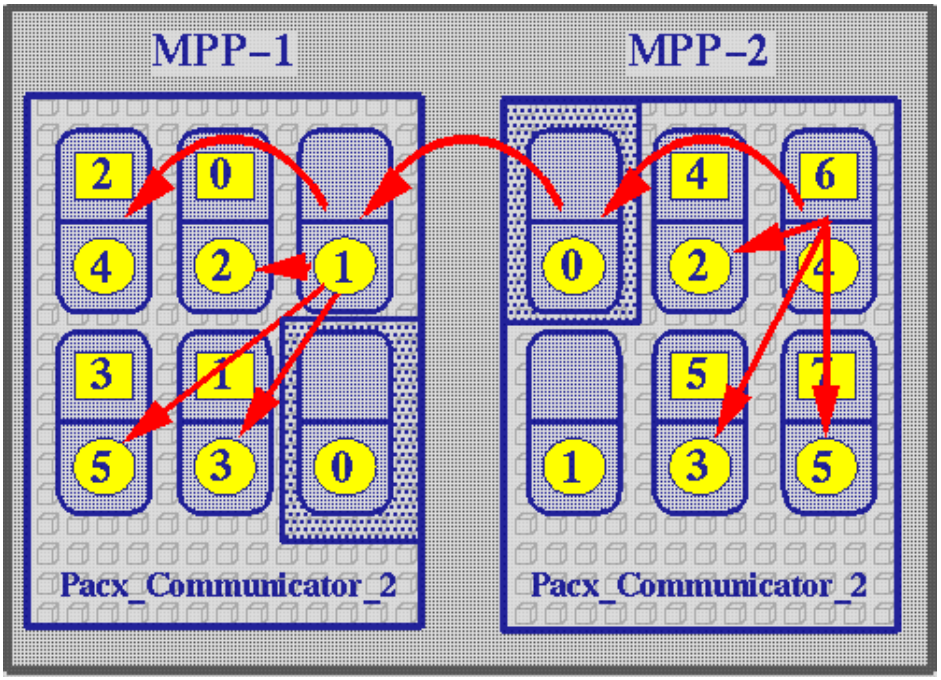


PACX provides an MPI_COMM_WORLD across the two machines involved. Numbers in the squares indicate node numbering as seen by the application in this communicator. If global node 6 wants to send a message to global node 2 the following steps are taken:

- Node 6 calls an MPI_Send specifying node 2 in communicator MPI_COMM_WORLD as destination.
- The MPI call is processed by the PACX library.
- The PACX call finds that node 2 is on the other machine.
- PACX splits the message into a command package and a data package.
- The command package contains all envelope information of the MPI call plus some additional information for PACX. The data package contains the data.
- Both packages are compressed to reduce network traffic.
- The outgoing communication node hands both packages over to the incoming communication node on the other machine using the socket connection.
- The incoming communication node receives the command package, uncompresses it, and interprets its content.
- It therefore combines the envelope received and the data and sends all that as a normal MPI message to the local destination given.
- Data are now safely delivered to the correct destination but the sender has no information about correct delivery.
- The incoming communication node hands back the return value of the MPI_Send it issued.
- The outgoing communication node hands this return value back to global node 6 as the return

value of the MPI_Send call initiated there.

For a global communication things become even more complicated. The next figure shows how a broadcast is handled correctly on two machines using PACX.



If global node 6 now wants to do a broadcast on MPI_COMM_WORLD the following steps are taken:

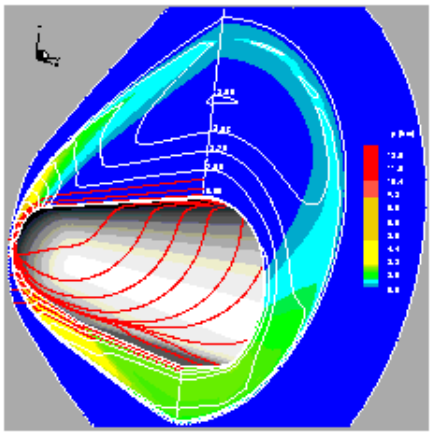
- Node 6 first sends a command package describing the broadcast and the data to be broadcasted to the outgoing communication node.
- It then does a broadcast in a communicator PACX_Communicator_1 especially provided by PACX to include all local application nodes.
- The outgoing communication node meanwhile hands the information over to the second MPP's incoming communication node.
- This node now sets up a normal MPI_Bcast from the command package and the data package and distributes it in a second communicator PACX_Communicator_2 provided by PACX including the incoming node and all local application nodes.

So far PACX has been installed and tested on an Intel Paragon and the Cray T3E. The most recent version supports usage of two MPP's but does not provide any data conversion. It has been tested to connect T3Es in Europe and the US succesfully. Future developments will include support for heterogeneity and support for usage of more than two machines. This will include handling the data conversion problem.

Application

The Navier-Stokes solver URANUS (Upwind Relaxation Algorithm for Nonequilibrum flows of the Univeristy of Stuttgart) was developed at the Institute for Space Systems at the University of Stuttgart. It is used for the simulation of nonequilibrum flows around reentry vehicles in a wide altitude-velocity

range. The unsteady, compressible Navier-Stokes equations in the integral form are discretized in space using a cell-centered finite volume approach. The inviscid fluxes are formulated in the physical coordinate system and calculated with Roe/Abgrall's approximate Riemann solver. Second order accuracy is achieved by a linear extrapolation of the characteristic variables from the cell-centers to the cell faces.



To compute large 3-D problems, the URANUS code was parallelized at RUS. Since the code is based on a regular grid a domain decomposition was chosen. This results in a perfect load balancing and an easy handling of communication topology. An overlap of two cells guarantees numerical stability of the algorithm.

So far the code is split into three phases (preprocessing, processing and postprocessing). Preprocessing is still done sequentially. One node reads in all data, does some preprocessing work, and distributes data to the other nodes. This may be a serious bottleneck when we simulate larger configurations and it will surely compromise parallel efficiency. Therefore, a future version based on a multiblock approach will provide parallel input. Experiments show us that this can dramatically reduce the startup phase of the code.

The processing part consists of setting up a heptadiagonal system of equations in parallel. The system is solved by applying a line relaxation method. Each tridiagonal system resulting from this is set up in parallel. The resulting overall system

$$A * x = b$$

is then split into two parts on each node: the part that is independent of all other nodes and the part that couples the local node with the others. So the matrix A is split into

$$A = L + M$$

where L is the local part and M the coupling part. An iterative procedure is used to solve locally. This allows the application to solve locally in parallel and exchange data only once per iteration.

To adapt the code for metacomputing the following steps are taken:

- Eliminate the bottleneck of reading in data on one node only. This will not only eliminate the I/O bottleneck, but since data must no longer be distributed to all nodes, network traffic can be substantially reduced.
- Reduce communication in the solver part by not updating the right hand side after each iteration. This may affect the solver and the stability of the method. Experiments have to be done to find a tradeoff between network traffic reduction and convergence speed.
- Extensively overlap communication and computation to reduce idle times for all processes.

Performance

The ultimate goal of metacomputing is to solve very large problems. Metacomputing provides more memory than is available on one machine and hopefully more performance. But while it is easy to double main memory by simply adding a second machine, the performance gained by the additional machine depends on the network connection. So when looking at performance for metacomputing, one has primarily to look at latency and bandwidth of the available network. Theoretical peak bandwidth of current networks can go up to 155 Mbit/s, but this may be available only for a short time window and may be rather expensive. Latency can not be reduced at will to compete with integrated systems. So overall performance for metacomputing will always depend on how much latency can be reduced.

In the following we want to give both first performance results for PACX and first results for URANUS. Since PACX and URANUS are far from being perfectly adapted for metacomputing across long distances, these results have to be seen as starting points rather than as final results.

For PACX four major questions have to be answered to get an estimate of how well the library performs:

- What is the overhead that PACX incurs in by adding a check for whether a communication is local or remote?
- What is the overhead that PACX imposes on a communication with respect to protocols, buffer copying, compression, and so on?
- What is the performance one can expect from PACX on a production network like the vBNS?
- What is the performance one can expect over a transatlantic connection that is not specifically dedicated to the application?

To answer these questions, one must have access to the resources required to do extensive testing. Experience has shown that it is rather easy to do testing on one machine and even on two machines that are in the vBNS. But working across the Atlantic Ocean is still a challenge. We have not yet had reasonable bandwidth between Stuttgart and the US, so those results are not available. Once a network connection is available, we hope to see similar results as those on the vBNS.

The following are results for latency, bandwidth, and time it takes to perform global operations with MPI on a single machine and using PACX between T3E's at PSC and SDSC. Furthermore we provide first results for the URANUS code if only for small test cases.

Latency

Latency is the time it takes for an zero-byte message to travel from one node to another and back divided by two. The latency overhead introduced by PACX to standard send calls of MPI for internal

communication is about 3 microseconds which is rather small. The additional latency incurred by accessing the TCP protocol stack and copying of data compares to latencies seen on workstations. One has to take into acccount that PACX actually sends two messages - one command message and one data message - so that the real latency is only ~2 milliseconds.

Latency		**Bandwidth**	
Mode	**microseconds**	**Mode**	**MB/s**
MPI	16	MPI	307
MPI+PACX	19	MPI+PACX	297
PACX internal	4000	PACX internal	0.25
PACX remote	40000	PACX remote	0.15

Bandwidth

Bandwidth as discussed here is aggregated bandwidth for very large messages. The overhead introduced by PACX before actually calling an MPI call for internal communication only slightly reduces bandwidth by about 3%. The low bandwidth for PACX itself seems to be due to inefficient usage of TCP buffering. This will be improved in future versions of PACX.

Global Communication

In transatlantic metacomputing it is time-consuming to synchronize an application. The following results therefore have to be seen as penalties that a code incurs if it makes use of synchronizing message-passing calls.

Barrier

Barrier synchronization is done rather fast on the T3E itself. PACX synchronizes both machines separately and then exchanges a synchronization message. After that message both machines can be sure that the other one has synchronized locally. Using a second barrier locally, they make sure that both machines run synchroneously.

Barrier Synchronization					
Mode	**number of nodes**				
-	4	8	16	32	64
MPI	2.5	3.5	2.7	3.3	3.4
PACX local	13300	13300	12900	13800	13400
PACX remote	157000	158000	156000	153000	172000

Broadcast

The algorithm for the broadcast operation as described above requires transmission of one message which imposes a latency on that operation.

Broadcast 1 Kbyte					
Mode	**number of nodes**				
-	4	8	16	32	64
MPI	26	38	49	61	71
PACX local	14800	14500	14700	21200	12800
PACX remote	159500	156600	170300	158700	158800

URANUS

The application tested is not yet adapted for metacomputing. It does no latency hiding and uses some collective operations. And due to the I/O bottleneck, results can not be expected to be spectacular.

In the following we give only the overall time that it takes to solve a small example (33K cells) and a medium sized (110K cells) one.

Small test case					Medium test case			
Mode	**number of nodes**				**Mode**	**number of nodes**		
-	8	16	32	64	-	16	32	64
MPI	47	32	22	17	MPI	94	59	46
PACX local	91	186	179	187	PACX local	291	293	303
PACX remote	-	218	217	268	PACX remote	513	527	-

Timings as given here include preprocessing, processing and postprocessing. Since it was difficult to do testing on more than one machine we calculated only 10 iterations. Normally it needs from 200 to 10000 iterations for the code to converge. But these first test results certainly point out the metacomputing challenges we face.

It is obvious that PACX imposes such a high overhead on the communication by using TCP that for the nonoptimized version of PACX and without having changed the code of URANUS we see a slow down for all problem sizes even if we are on the same machine. However, the message that we see from these first results is that timings remain nearly constant which implies that it is latency that slows down the calculation. If we then go to two machines, we see an additional slow down and again nearly constant values for timings. Again it seems that latency dominates the results.

Conclusions

We have shown that two T3E's can be successfully coupled even across the Atlantic Ocean.

We have shown that PACX works correctly, and the concept can be extended to more than two machines. PACX will be improved by reducing latency, and latency can be hidden using asynchronous communication for the I/O nodes.

Uranus has been shown to scale well on large numbers of nodes. Improvements will include reduction of communication in the solver and elimination of the I/O and preprocessing bottleneck.

Metacomputing is latency-bound for this application. Other applications that are interesting to explore are coupled simulations that steer each other or exchange data, such as ocean-atmosphere modeling.

References

[1] Edgar Gabriel, Thomas Beisel, Michael Resch. **Erweiterung einer MPI-Umgebung zur Interoperabilitaet verteilter MPP-Systeme.** RUS-Report 37, January 1997.
[2] H.-H. Fruehauf, O. Knab, A. Daiss, U. Gerlinger. **The URANUS code - an advanced tool for reentry nonequilibrium flow simulations.** Journal of Flight Sciences and Space Research (19)

219 - 227. 1995.
[3] T. Boenisch, A. Geiger **Implementierung eines 3-D Stroemungscodes auf Parallelrechnern.** RUS-Report 37, April 1996.
[4] A. Wierse **Performance of the COVISE visualization system under different conditions** in Visual Data Exploration and Analysis II, Georges G. Grinstein, Robert F. Erbacher (Eds.), Proc. SPIE 2410, pp.218-229, San Jose 1995.

Author Biography

MICHAEL M. RESCH is currently working with the Parallel Computing Group of the High Performance Computing Center at Stuttgart were he coordinates the activities in the metacomputing project G-WAAT. He got his Dipl.-Ing. in Technical Mathematics from the Technical University of Graz/Austria. He started working with clusters of workstations and is currently involved in European projects related to high performance computing and networking.

- mailto: resch@rus.uni-stuttgart.de
- URL: http://www.uni-stuttgart.de/pcg/pcg/resch/resch.html

BRUCE LOFTIS has a long history in high-performance computing and now holds the position of MetaComputing Project Leader at the Pittsburgh Supercomputing Center. He earned B.S. and M.S. engineering degrees from the University of Texas and a Ph.D. in water resources engineering from Colorado State University. He has held faculty positions in Civil Engineering at Colorado State University and in Operations Research at North Carolina State University. Research interests include environmental modeling, parallel numerical methods, large-scale mathematical optimization, and large-scale distributed applications.

- mailto: bruce@psc.edu

RAGHU REDDY is a Computational Scientist at PSC and has been working with researchers in optimizing their programs for supercomputers. He has an MS in Agricultural Engineering from Colorado State University and has been working in the high-performance computing platforms for more than ten years, with experience ranging from vector processors to the MPP’s. Research interests include parallel finite elements, parallel numerical algorithms, and distributed computations.

- mailto: rreddy@psc.edu
- URL: http://www.psc.edu/~rreddy

Acknowledgements

The authors gratefully acknowledge support from their home organizations RUS and PSC and supercomputing time provided by the San Diego Supercomputing Center.