Forschungszentrum Jülich

Zentralinstitut für Angewandte Mathematik

Interner Bericht

Metacomputing in Gigabit Environments: Networks, Tools, and Applications

Thomas Eickermann, Jörg Henrichs, Michael Resch, Robert Stoy*, Roland Völpel***

FZJ-ZAM-IB-9824

FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Metacomputing in Gigabit Environments: Networks, Tools, and Applications

Thomas Eickermann, Jörg Henrichs, Michael Resch, Robert Stoy*, Roland Völpel***

FZJ-ZAM-IB-9824

August 1998

(letzte Änderung: 14.08.98)

Parallel Computing 24 (1998), pp. 1847-1872

(*) Höchstleistungsrechenzentrum Stuttgart, Abteilung Parallel Computing, Allmandring 30, D–70550 Stuttgart, Germany
(**) GMD – Forschungszentrum Informationstechnik Schloß Birlinghoven, D–53754 Sankt Augustin, Germany

Metacomputing in Gigabit Environments: Networks, Tools, and Applications

Th. Eickermann[a*] , J. Henrichs[a], M. Resch[b†], R. Stoy[b], and R. Völpel[c]

[a]Central Institute for Applied Mathematics, Forschungszentrum Jülich, D–52425 Jülich, Germany

[b]Höchstleistungsrechenzentrum Stuttgart, Abteilung Parallel Computing, Allmandring 30, D–70550 Stuttgart, Germany

[c]GMD – Forschungszentrum Informationstechnik, Schloß Birlinghoven, D–53754 Sankt Augustin, Germany

This article gives an overview over recent and current metacomputing activities of the Computing Centers at the Forschungszentrum Jülich, the GMD Forschungszentrum Informationstechnik and the University of Stuttgart. It starts with a discussion of the underlying network connections which are dedicated testbeds. A library that provides an MPI–API for metacomputing applications is presented, as well as a library that supports load–balancing and latency hiding. Finally, results from several applications ranging from tightly coupled homogeneous to loosely coupled heterogeneous metacomputing are presented.

keywords: Gigabit Networking, Transatlantic Metacomputing, MPI, Load Balancing, Heterogeneous Metacomputing

1. INTRODUCTION

During the last years metacomputing has become a catchword among the supercomputing community. Like other catchwords it is mostly unclear what it is supposed to mean. However, commonly it describes some sort of linking together of computational resources that compete with supercomputers or try to outperform them, at least theoretically. This development is driven mainly by two ideas:

First, supercomputing resources are expensive and have a short life–cycle. They should be shared between different research centers for economical reasons. Such resources not only include supercomputers of different architectures (massively parallel and vector–based), but also high quality visualization hardware like the CAVE [1] and other devices that produce or consume data at high rates. An example for the latter are Magnetic Resonance (MR) Tomographs. Combining these resources leads to a heterogeneous meta-

*Email: Th.Eickermann@fz-juelich.de
†Email: Resch@hlrs.de

computer. Typical examples for such a scenario are the coupled simulation of groundwater flow and transport of contaminants in the groundwater and the real-time visualization of brain activity as described in this paper.

The second idea is that the coupling of supercomputers offers a way to increase the peak performance of a machine. In principal two T3Es could be twice as powerful as one. Typical applications for such a homogeneous metacomputing–scenario are Monte Carlo codes as described in this paper.

Since the efforts that have to be made to couple such powerful machines is high and the benefit is limited to very loosely coupled applications it is obvious that metacomputing as described here is restricted to a limited number of special applications. However, those applications can then benefit substantially from the accumulated performance of a metacomputing environment.

Metacomputing currently faces a number of problems. Some of which are well understood, others still have to be investigated thoroughly. Following a layered approach the problems are threefold.

First, there is the network problem. Coupling of remote resources requires fast and reliable networks. A resource like the internet is not designed to support the traffic characteristics of a metacomputing application. A software relying on the internet may therefore sometimes yield acceptable results and sometimes fail completely. One of the prerequisites for metacomputing is therefore to be able to provide the application with a stable and fast network connection that can be dedicated to a single application run. Typically such quality of service can be provided by ATM. But, so far, ATM–networks for research activities are not yet widely available.

Second, there is the communication problem. While each hardware vendor has adopted the MPI standard and provides his users with fast and stable implementations, there is no support for metacomputing. Since even the MPI–Forum has refused to put the topic on its todo list it is up to the user to find ways how to overcome the problem. PVM definitely is designed to overcome that problem. But then PVM is no longer the standard in the field and most users have moved to MPI and do not want to change their code for metacomputing experiments. A tool to bridge the gap between PVM and MPI would be PVMPI [9]. But again this would require the user to substantially change his code. It has therefore become necessary to set up tools that provide the user with a global MPI — often called an interoperable MPI. One such tool, PACX–MPI, is described in this paper.

Third there is the application level at which one has to consider the limitations of metacomputing. Latencies even on fast networks tend to go up to several milliseconds. Even traveling at the speed of light a message traveling from Germany to the US will take about 25 milliseconds. And even though bandwidths are constantly increasing it is unlikely that external bandwidths will ever be able to compete with the internal bandwidth of a highly integrated MPP. Applications have therefore to take into consideration a substantially higher latency for communication between machines. And in addition they will have to deal with the problem of bandwidths that vary by orders of magnitude between internal and external communication. An approach to overcome this problem is latency hiding by overlapping communication with calculation. Since it is often a non–trivial task to incorporate such methods in an application, supporting libraries would be useful. Such a library is described in the load balancing section of this paper.

Besides these technical problems there is a number of organizational ones. Running metacomputing applications requires a synchronization the computing resources of several computing centers. Furthermore, the input and output data have to be distributed and collected. This is not a real problem as long as metacomputing is performed on an experimental basis. But in a production environment a secure and consistent access to distributed computing resources and data will be essential too. Research projects that address these topics include UNICORE [2] and HPCM [3].

In the following, our current activities and results in the above mentioned aspects of metacomputing: networking, tools and applications will be discussed.

2. NETWORKS

A key factor for the success of metacomputing activities are communication networks that provide high–bandwidth and low–latency connections between the components of the metacomputer. Generally, the performance available over wide area networks is low compared to the communication within a parallel computer.

In Germany, the network that connects research, science and educational institutions with each other and the rest of the internet is operated by the DFN–Verein, an association of these institutions founded in 1984. Since 1996 this network is based on ATM–technology and allows for access capacities up to 155 Mbit/s. Plans exist to extend the bandwidth into the Gbit/s range, on a national basis, in the near future. To prepare this transition, two testbeds have been set up in the western and southern parts of Germany. They will serve to evaluate new network technology as well as to gain experience with applications requiring bandwidths beyond the currently available 155 Mbit/s. In the area of scientific computation, such applications can e.g. be found in multimedia, distributed access to huge amounts of data and of course in metacomputing, which is the subject of this article.

Besides providing Gigabit networks on a national scale, there is also a requirement to interconnect supercomputers across national and continental borders. Currently, speeds in the range of several Mbit/s are possible. The German National Research Network B-WiN provided by the DFN connects the to US with a bandwitdth of 2*45 Mbit/s. This bandwidth is shared by all research institutes within the DFN community. The long term performance requirements of a metacomputer, especially regarding packet losses and delay, can currently not be met by this standard Internet path. Therefore, a separate ATM-Link between the Rechenzentrum Universität Stuttgart (RUS) and the vBNS, the network that connects the US national Super Computing Centers, has been established.

2.1. Gigabit Testbed West

The first of the two German testbeds started in August 1997. It is a joint project of the Forschungszentrum Jülich and the GMD–Forschungszentrum Informationstechnik in St. Augustin close to Bonn. In the beginning the two locations — which are approximately 100 km apart — are connected by an OC–12 ATM–link (622 Mbit/s) based upon Synchronous Digital Hierarchy (SDH/STM4) technology. The connection is provided by o.tel.o Service GmbH and uses the optical fiber infrastructure inside the power lines of the German power supplier RWE AG. Fore Systems ATM switches (ASX–1000) link the OC–12 line to the local ATM networks of the research centers. The link will be upgraded to OC–24 (1.2 Gbit/s) and OC-48 (2.4 Gbit/s) as soon as the next generation of switches

is available. This is expected for mid 1998.

For the purpose of metacomputing the CRAY supercomputer complex in Jülich, consisting of two T3E massively parallel computers and a T90 vector–computer, is connected to the IBM SP2 parallel computer in St. Augustin via the OC–12 line. Whereas networking components with gigabit capabilities are already available or will be available within the next few months, connecting the CRAY and IBM supercomputers to such a network imposes more problems.

The most convenient and simple solution — single ATM interfaces supporting OC–12 or more — will not be available before late 1998, if at all. Currently several OC–3 ATM interfaces are installed in the machines in Jülich and St. Augustin. One possible way to increase the bandwidth is to multiplex the communication of a single application over more than one interface. This is a challenge left for the application (or the underlying communication library) alone since there is no support by the operating systems.

Still there are other options to choose from. A well–established networking technology is the 'High Performance Parallel Interface' (HiPPI). Although its peak bandwidth of 800 Mbit/s can only be achieved with very large transfer–blocks (1 MByte) and a low–level protocol, the so–called framing–protocol, reasonable results can be obtained even with IP communication. In Jülich, HiPPI is used to interconnect the CRAY machines. Transfer rates of more than 400 Mbit/s can be seen for single socket–based connections. An ATM/HiPPI–gateway by Ascend Communications allows to connect an 800 Mbit/s HiPPI network to a 622 Mbit/s ATM network for IP communication. It is also possible to tunnel the HiPPI framing–protocol through the ATM connection if there is another gateway and a HiPPI device at the other end of the line.

For the IBM SP, a so called 'SP Switch Router' has recently been announced. It will directly connect the SP–internal High Performance Switch to an external router with a peak bandwidth of more than 1 Gbit/s.

Several SUN workstations and servers in Jülich and St. Augustin are equipped with OC–12 ATM interfaces and are connected via the Gigabit Testbed. Preliminary measurements with this equipment show a high reliability of the network and reasonable throughput. Message latency of TCP socket–connections is below 1 msec. This is consistent with the average latency of 10 msec/1000 km of the SDH/ATM–line as specified by the provider o.tel.o. The measured transfer rates for this connections are in the range of 160 Mbit/s which is well below the theoretical limit for IP connections over an OC–12 line. Measurements with two pairs of workstations, however, result in an aggregate bandwidth that is almost twice as high. This shows that the bottleneck are neither the ATM/SDH line nor the ATM switches in the path but the networking capabilities of the workstations. It should be noted, that the measurements have taken place while the participating servers were under regular workload. Using dedicated machines and tuning their kernel networking parameters is currently under way and should lead to better results.

2.2. Transatlantic Network Connections

During Summer 1997, in cooperation with RUS, the Pittsburgh Supercomputing Center and the participating network providers, a transatlantic Metacomputer was established connecting the two CRAY T3Es across the Atlantic through a separate dedicated ATM Channel with a 2 Mbit/s link. For the Supercomputing '97 event, this network was

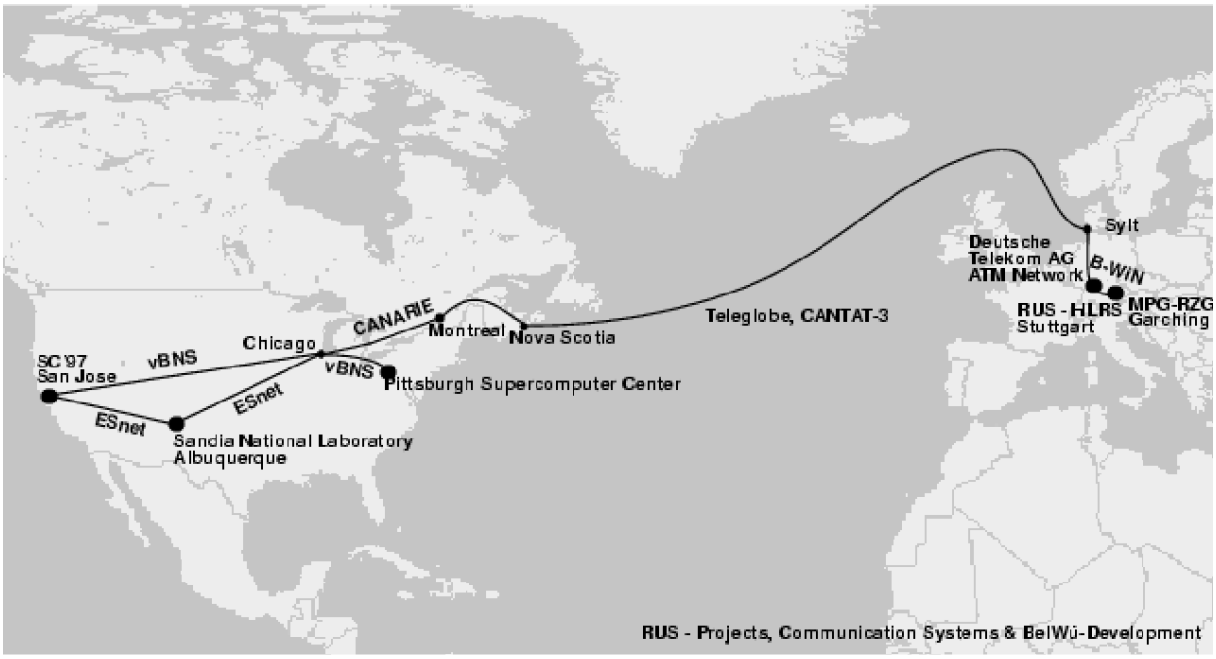

Figure 1. Network for transatlantic metacomputing demonstrations during SC'97

extended to Sandia National Laboratory, Albuquerque New Mexico, and to San Jose. Figure 1 shows the the geographic extension of the transatlantic metacomputing environment during SC' 97.

The Max-Planck-Institut-für-Plasmaphysik, Garching, used the transatlantic ATM-Link for their demonstrations at SC'97. They werde connected to the dedicated ATM-Link end-point in Stuttgart through B-WiN.

2.2.1. Network Performance Issues

In the given context, the most relevant performance parameters were latency and bandwidth. With respect to latency, comparing the cross atlantic standard path provided by DFN and the dedicated ATM-Link, it was interesting to note the effect due to the number of routers involved and the translation of packet losses into additional delays. The results achieved on the network connection between a test workstation at RUS and the CRAY T3E in Pittsburgh over the Standard Path and the dedicated Link are depicted in Table 1.

The network performance of the standard path is strongly influenced by the European working hours. During night-time, the packet loss and packet round-trip time were acceptable and a TCP throughput of approx. 250 kByte/s was achievable. However, during the daytime, the IP packet losses (with packetsize of 1 kByte) downgraded the TCP througput to less than 50 kByte/s. The mean packet round-trip time on the standard path ranged from 150 to 300 ms.

On the dedicated ATM-Link, there were practically no packet losses (during SC'97 a small number of packet losses appeared during the change over from CANARIES ATM-

Table 1
Comparison of network performance on DFN's standard internet path vs. direct ATM Link. [1]average value (variation between 160 and 300 ms). [2]variation between 150 and 155 ms. [3]the socket buffer used was 64 kB.

Connection	Bandwidth [Mbit/s]	no. of routers	tcp-throughput [kByte/s][3] day/night	packet losses [%] day/night	delay [ms] day/night
DFN	2*45	15	50/300	30/3	180[1]/160
ATM-Link	2	5	200/-	0/0	150[2]/-

network to CA*Net II) with a nearly constant round-trip time of 150 ms. This good link performance resulted in a constant TCP throughput of 200 kByte/s, which is the maximum throughput available on a 2 Mbit/s ATM-Link.

The higher number of routers on the standard path introduced a relatively small latency, so in the case of a small load as seen during European night-time hours the round-trip time on the standard path is comparable to that of the direct ATM Link.

Figure 2 shows a comparison of the network delay and packet losses during a 24 hour period over the standard path and the dedicated ATM-Link. The data on the dedicated ATM-Link was captured during SC'97, the data on the standard path some time after SC'97.

As is well known, the TCP performance on links with large bandwidth times delay products is strongly dependant upon the TCP Window size, which is configured on the end-systems through the TCP socket buffer sizes. The following figure shows that on the 2 Mbit/s ATM-Link a socket buffer size of 64 kByte is required for maximum TCP throughput.

2.2.2. Political Issues

Using the vBNS, as shown in Figure 3 above, required the formal approval of the National Science Foundation. The SC'97 preparation and demonstration was the first involving a non-US center accessing the 'project only' network infrastructure. At the same time, useful discussions were conducted between the DFN and the NSF regarding the global context and placement of the STAR TAP (Science, Technology and Research Transit Access Point) in Chicago as an international exchange point and its geographical relation to the DFN and Europe's Point-of-Presence close to Washington DC.

3. TOOLS

3.1. PACX–MPI

PACX–MPI was developed to allow to extend MPI [4] communication beyond the boundary of an MPP system. Typically, on such systems an optimized version of MPI is offered that does not allow to communicate outside that system. Only recently commercial implementations have come up that allow to run one single MPI application across a series of machines. But then again the user is restricted to one hardware vendor [5,6]. Public domain implementations of MPI like MPICH [7] support clusters of machines but can

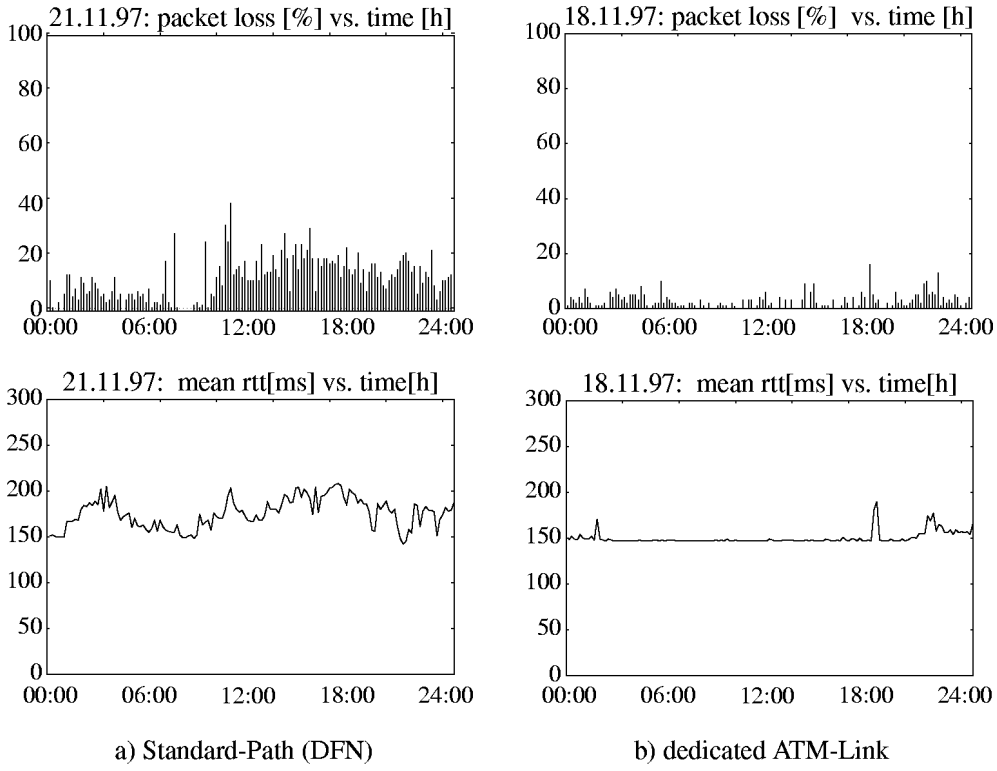

Figure 2. Round trip time and packet loss on the standard path and the direct ATM Link during a 24 hour period.

not be used to couple two or more MPPs efficiently. Another option for metacomputing would be to use PVM [8] but since MPI has become a standard for MPP programming most users have migrated from PVM to MPI already.

There is a number of projects that aim to support the MPI programmer in metacomputing but all of them require changes in code [9–11]. One of them is PVMPI [12]. PVMPI focuses specially on establishing a connection between two already running MPI applications. Whenever an MPI application starts up it creates a communicator MPI_COMM_WORLD. This communicator can not be extended to add new processes. MPI-2 will be able to do so, but so far dynamic process control is not available. Furthermore, MPI has no defined way to talk to external programs. PVMPI provides a way to overcome these restrictions. By incorporating some parts of PVM into MPI the user is given the opportunity to create an intercommunicator from the two separate MPI_COMM_WORLDs of the two applications running. On this newly defined intercommunicator point-to-point communication is possible.

The disadvantages of that concept with respect to our intended project are twofold. First, we would have to add some non-MPI calls to the application, making it impossible to run the same code on one or two machines at the highest possible performance. Second, this would have meant to redesign the code in order to reduce the communication

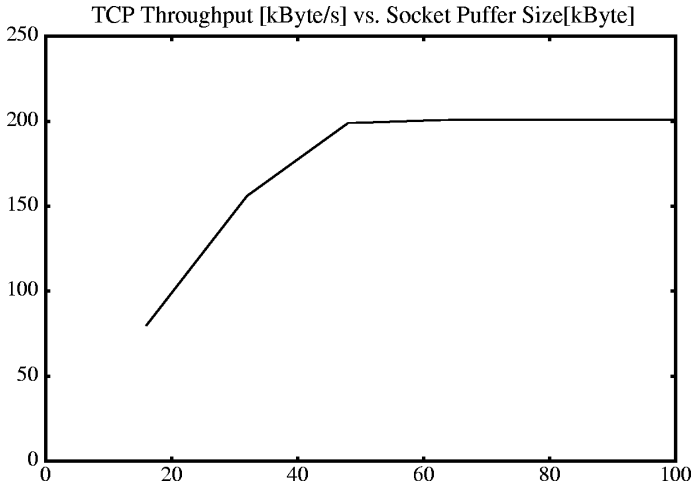

Figure 3. TCP throughput on the transatlantic 2 Mbit/s ATM Link as a function of endsystem TCP socket buffer size.

between processes running on different machines to simple send/recv communication. We therefore decided to implement PACX–MPI as a transparent extension to existing MPI implementations.

The first version of PACX–MPI evolved from a project aiming to exploit at the same time the advantages of an MPP system and of a vector-supercomputer. Some parts of the code used in that project performed excellently on an Intel Paragon. They were well fit for a distributed memory system with a high speed network. Other parts only performed well on a Cray YMP vector-supercomputer. The idea behind the initial project was therefore to let the parallel parts of the program be run on the distributed memory machine while the vector parts would run on the vector machine and both could communicate in one MPI world [13]. At that time no MPI implementation was available that would have allowed to integrate the two machines into one single environment and at the same time exploit the performance of the fast network of the Intel Paragon. Later on this concept of connecting a vector supercomputer as an additional processor to an MPP was extended to couple two or more MPPs in one single computing resource [14,15].

The goals of these projects were the following:

- Provide the user with a single virtual machine on which MPI applications can be loaded. No changes to the code are necessary.
- Use highly tuned MPI for internal communication on each MPP.
- Rely on fast standard communication protocols for external communication.

3.1.1. Basic Concept

The concept of PACX–MPI is based on two major design decisions:

- Design PACX–MPI as an intermediate layer between the application and the communication protocol.
- Let all external communication be done by two servers that are responsible for incoming and outgoing traffic respectively.

There are some advantages and some disadvantages in that concept that should be briefly discussed here. Using two servers that have to handle all communication between the MPPs involved means to implement a bottleneck for all external communication. However, it makes it easier to handle all external traffic by reducing the number of connections that have to be supervised. Furthermore, one has to take into account that in most scenarios a bottleneck is already constituted by the fact that physically there is only one connection available and that all external communication has to be routed through one I/O node of the MPP.

The layered approach seems to be the most flexible solution to reach the goal of highest performance on both internal and external communication. Although one has to implement all MPI calls as separate PACX–MPI calls one may very easily decide about communication methods without having to touch either the local MPI implementation or the remote communication software of the two servers. Furthermore, such a layered approach allows to easily optimize global communication and global operations that otherwise may harm the performance of an application running in a metacomputing scenario.

3.1.2. Implementation

PACX–MPI has been implemented in C. For external communication standard TCP/IP protocol is used [16]. However, there is also a version of the interface available that is based on HiPPI. Like MPI, PACX–MPI has language bindings for FORTRAN 77 and ANSI C. But while MPI consists of more than 120 function calls PACX–MPI was restricted to a smaller number. It mainly implements those functions that are the most frequently used ones and omits the ones that are not that important for numerical simulations. At this time PACX–MPI supports the following calls:

- Initialization and control of the environment
- Standard point-to-point communication
- Collective operations:
  MPI_Barrier, MPI_Bcast,MPI_Reduce and MPI_Allreduce
- Standard nonblocking communication

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.

Point-to-point Communication:

The handling of a point-to-point communication is shown in the next figure. Since PACX–MPI provides an MPI_COMM_WORLD across the two machines involved there has to be a mapping of local process numbering and global one. Numbers in the squares

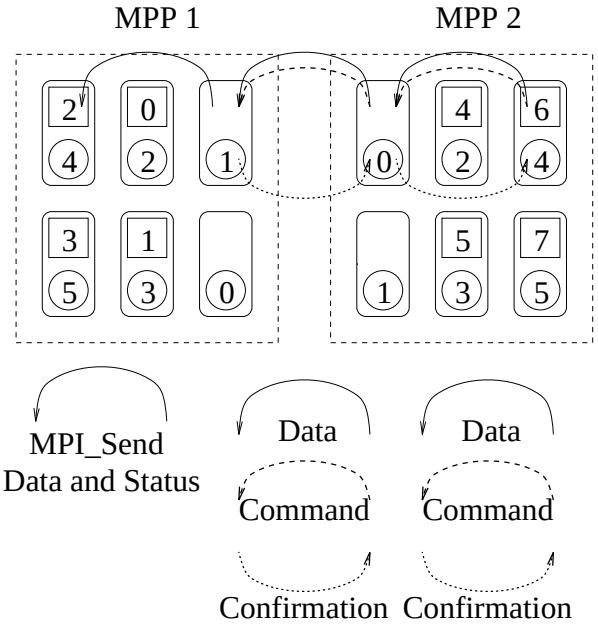

Figure 4. Point-to-point communication between two MPPs using PACX–MPI.

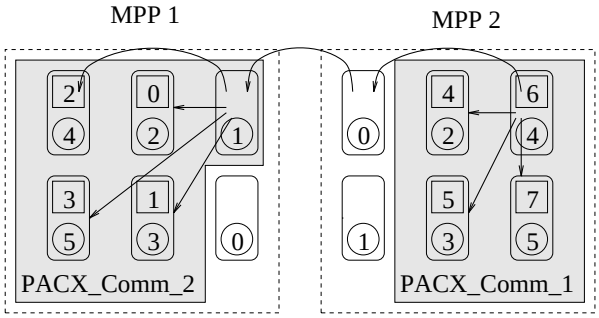

Figure 5. Broadcast operation on two MPPs.

indicate this global node numbering. 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. This call is processed by the PACX–MPI library.
- PACX–MPI finds that global node 2 is on the other machine. So it has to hand the message over to the PACX-server . For this the message is split into a command package and a data package. The command package contains all envelope information of the original MPI call plus some information for PACX–MPI.
- Both packages are compressed to reduce network traffic and sent over to the other systems incoming communication node. There, data is uncompressed and the command package is interpreted.
- Using this information a normal MPI_Send to the destination node is issued. The return value of this call is handed back to the first system to be handed back to the original sender.

Global Communication:

For a global communication things become even more complicated. The next figure shows how a broadcast to MPI_COMM_WORLD from root 6 is handled correctly on two machines using PACX–MPI.

The following steps are taken:

- Node 6 first sends a command package describing the broadcast and the data to be broadcast to the outgoing communication node.
- It then does a broadcast in a communicator PACX_Comm_1 especially provided by PACX–MPI 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_Comm_2 provided by PACX–MPI including the incoming node and all local application nodes.

This concept for global communication allows to overlap communication to the second MPP and internal communication. Furthermore, the local broadcast communication on the two machines is done asynchronously.

3.1.3. Performance

Some measurements of performance of PACX–MPI have been done for standard benchmarks like the ping-pong test [17]. Those tests were performed in the frame of a testbed in which a T3E at Stuttgart and a T3E at Pittsburgh were interconnected by a dedicated 2 Mbit connection. The results for latency are as follows:

- The overhead incurred on internal communication by the layered approach is about 3 microseconds. Latency for internal communication is increased from about 16 microseconds to about 19 microseconds.
- Latency incurred by usage of the TCP/IP protocol is in the range of 4 milliseconds. This compares well to results known from usage of the TCP/IP protocol in clusters of workstations.
- The latency across the transatlantic connection is in the range of 75 milliseconds where 70 milliseconds are imposed by the network.

Also for bandwidth acceptable results can been seen so far:

- The overhead incurred on internal communication only reduces bandwidth by about 3 percent from 307 MB/s to 297 MB/s for Cray's MPI.
- Bandwidth on the transatlantic network connection goes up to about 1 Mbit/sec which is acceptable.

3.1.4. Applications

Since PACX–MPI is developed to meet the needs of special applications all development was thoroughly tested with respect to the performance that our applications could achieve. Results for two applications will be given below. They show clearly that usage of PACX–MPI on a very low bandwidth network limits the range of applications to a small subset. The main focus will be on loosely coupled applications that are able to hide latencies in the range of tens of milliseconds. As network bandwidth on wide area networks is growing

bandwidth on the other hand will not become a critical issue for metacomputing. The first test results however show that metacomputing is worth doing and that PACX–MPI can substantially contribute to that.

3.1.5. Future Work

Experiments have shown that the choice of protocol may severely influence the latency and bandwidth that can be achieved. Since in the future fast network connections will be based on ATM, it is planned to base external communication of PACX–MPI on ATM directly rather than using TCP.

Furthermore, we will have to extend the functionality of PACX–MPI to be able to support a wider range of applications. The user should however be aware of the fact that a metacomputing application will only perform well if sophisticated functions of MPI that require tightly coupled processors and fast networks are avoided.

Currently PACX–MPI has been installed on an Intel Paragon, a Cray T3E and an IBM SP2. A version for a Hitachi SR2201 is in work. In a next step PACX–MPI will be ported to machines of the NEC SX series to be able to couple those vector computers to an MPP. Furthermore, PACX-MPI is in use by several research centers in Germany and the USA that provide feedback for further development.

3.2. Optimizing and Load Balancing Metacomputing Applications

Practical applications often fail to utilize the potential of metacomputers. Using the available memory is no particular problem [17], and also the distribution of different programs between different computers, each one being a part of a larger application, was done successfully [21]. But it is still quite difficult to get a speed-up when distributing one, possible tightly-coupled code. Even when coupling different programs, it might be desirable to further speed-up one of the programs (e.g. to get a better load balance) by using processors available on the other machine. So while we will only examine tightly-coupled, single applications in this section, the ideas apply to a broad range of metacomputing applications.

Our first experiments were done in the Regional Testbed NRW [22,23], were we tried to distribute a tightly-coupled application across a metacomputer (see chapter 4.1). But in this application we not only got a slow-down when remote processors were added, even the metacomputer itself did not scale: the run time increased, when further processors were added [22]. This difficulty has been observed in many metacomputing projects, see e.g. [17,24,25].

The reasons for this unexpected behavior are manifold: First of all, the high latency of the external connection slows down the algorithm significantly. Most algorithms are tuned for a homogeneous machine and a fast interconnection network. Therefore no attention is spent to latency-hiding, which results in a general slow-down if some of the interconnects, the external ones, are slower than the internal ones. Existing message-passing libraries like MPI [4,7] or PVM [26] actually hide this kind of heterogeneity. While this makes developing parallel programs easier, it makes it difficult to optimize communication, for example by combining several messages into only one external message. This is of course due to the lack of application-specific information, they do not know which messages can be combined without causing a deadlock. For the programmer it is quite a lot of work to perform this straight forward optimization.

Another reason for the performance loss is the comparatively low bandwidth of external connections. The performance of modern ATM connections seems to be comparable with the internal communications: the Cray T3E offers a bandwidth of approximately 270 MByte/s (using MPI), the SP-2 installed at the GMD offers a bandwidth of approximately 80 MBytes/s and the ATM connection between Jülich and the GMD has a theoretical peak performance of about 77 MByte/s. But this comparison is misleading: an MPP system offers this bandwidth between a large number of PEs at the same time which enables an application to exchange more than 138 Gigabyte of data per second on a 512 processor T3E. For a real application this 'overall'-bandwidth must be considered to get an idea of the influence which the external communication will have.

The third problem is load balance. Even when using two computers of the same kind, there are often speed differences due to different compiler or library versions, different amounts of memory per processor, sometimes also different hardware (clock speed etc.). On the one hand, this is one of the great advantages of metacomputing: if an application shows an inherent static load imbalance, it can be easy to get a much better load balance by running some processes on a faster computer. In [22] an example is given, where the runtime of a program was reduced by a factor of nearly two simply by placing one process on a node of the SP2 instead of the Intel Paragon. On the other hand, a previously load-balanced application may be imbalanced on the metacomputer.

We therefore looked for a way to optimize an existing application without changing the underlying algorithm or forcing too many changes to the application itself. The most important point is to optimize the external communication by combining external messages between the machines to one message, so that we loose only time for one startup [22]. If possible, we would like to overlap communication and computation, so that even this startup time can be hidden.

We are trying to reach these goals by using a specialized library, which will have more information about the communication structure of the program than a general message-passing library and use these data to improve the performance. We assume a simple structure of a parallel program like the one suggested in [27], which is typical for a wide variety of programs. In this model each process performs three different kinds of operations (for simplicity we are neglecting the initialization and clean-up phase here):

- Local computations involving no communication. All processes work in parallel without having to exchange any data or synchronize.

- Neighbor communication: in this case, data are exchanged with 'neighbor' processes. This requires a synchronization with the neighbor processes, but no global synchronization.

- Global communication; this involves a synchronization of all processes.

Assuming this parallel program structure, the only application-specific data the new communication library needs is the communication structure, i.e. which processors are considered to be neighbors and therefore exchange data in the neighbor communication phase. Having this information available, the library can multiplex messages which are sent to the same remote machine by combining them into one, thus reducing the number of ex-

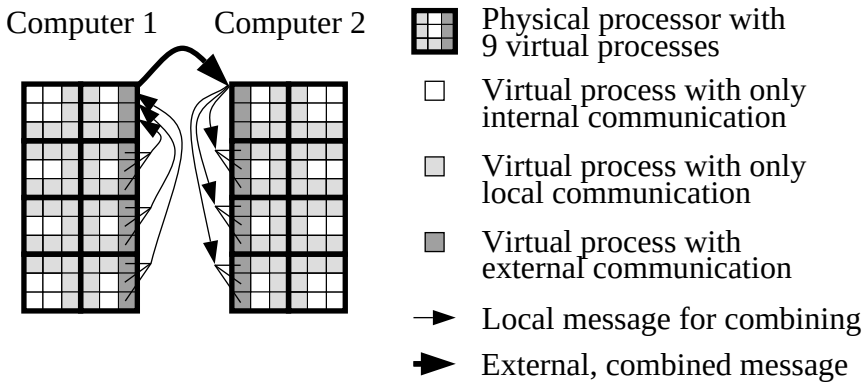

Figure 6. Scheduling using virtual processes

ternal messages. This combined message is distributed ('demultiplexed') to the proper receiving processes by one process on the remote machine.

To enable the overlap of computation and communication without having to change the algorithm we use the concept of 'virtual processors' or 'virtual processes' [28,29]. This means that the application is distributed on more processors than physically available. Each physical processor gets a number of virtual processes and has to schedule them for computing. The advantage of virtual processors are twofold: Firstly, by an appropriate scheduling it is possible to perform some latency hiding. Secondly, they can be used for load balancing purposes. In Figure 6 an example is shown, in which 8 processors on each of two machines form a metacomputer. By using 9 virtual processes on each machine, you have up to three different kind of virtual processes on each processor: processes which have to communicate with a remote processor, processes which only have to do local (meaning within the same machine) communication, and processes which only have to communicate with processes on the same processors. By an appropriate scheduling, i.e. first execute the processes, which have to use external communication, then processes, which will communicate locally and at last start the remaining processes, it is possible to overlap the communication time with computation of the remaining processes. Of course, this will increase the number of messages (while reducing the size of each message) within the system, since more processes have to exchange data. As a first optimization, messages which are sent to the same receiving processor are combined into one message, which is sent as soon as all virtual processes, which have to send data to the receiver, have computed their new values. This ensures that not more messages are sent by using virtual processors than by using only one process on each processor[3]. Furthermore, external messages, messages sent to a virtual process on a different machine, are sent to one 'collector'-process on the same machine, which in turn combines these messages and sends only one external message to a remote machine. On the receiver side, this message is demultiplexed and then locally distributed to the proper receiving processors which in turn take these messages apart, so that the messages for each virtual process is produced.

Virtual processes can also be used to improve static load balance. By increasing or

[3]Internal messages, a message for a virtual process on the same processor, remain within the memory of a processor so that no additional cost is involved.

decreasing the number of virtual processes on a processor the different speeds of the processors can be compensated for.

3.3. Implementation

There are different ways to implement virtual processes:

- Run several processes on each processor, each realizing one virtual process. The disadvantage of this implementation are the high cost for context switching and the additional operations for exchanging internal messages.
- Generate several threads on each processor, one for each virtual process. Since threads are not available on the Cray T3E this possibility was not considered. Additionally one has to deal with the differences between thread libraries on different machines.
- Use only one process (and one thread) on each processor and let this process simulate the different virtual processes. The additional benefit of this implementation is the easy and platform-independent way of scheduling the virtual processes. Otherwise, changes to the operating system or thread libraries would be necessary to implement the intended scheduling, which in turn would be a major obstacle in a metacomputing environment due to incompatibilities across the different machines.

We decided to implement the virtual processes by simulating them within one process. A prototype of this library, programmed in C++, is being developed and tested.

3.4. Results

We used the library for the algorithmic kernel of a partial differential equation solver, a multigrid code which solves a system of linear equations with of 2.3 million variables. Figure 7 shows our first results, where we used the same number of processors on each machine (so '8' processors means, that we used 4 processors on the Cray T3E, and 4 processors on the IBM SP2). We compared a 'standard' distribution, using one (virtual) process on each processor, with a distribution, which used 16 virtual processes on each processor. Not only was the optimized implementation faster than the original version (as much as ten times on 32 processors), it was also possible to get a speed-up by adding additional processors to the metacomputer. This experiment was done without paying attention to load-balancing, each SP2 processor had the same amount of work to do as a T3E processor. As soon as the static load-balancing scheme is enabled, we expect to get even better results.

4. APPLICATIONS

4.1. TRACE/PARTRACE

The program TRACE (Transport of Contaminants in Environmental Systems) simulates the flow of water in variably saturated, porous, heterogeneous media. It is used in combination with the program PARTRACE (PARticle TRACE) for 3–D simulations of particle transport in ground water [30]. The programs have been developed at the Institute for Petroleum and Organic Geochemistry at the Forschungszentrum Jülich. TRACE is based upon 3DFEMWATER, a ground water simulation code by Yeh [31]. PARTRACE

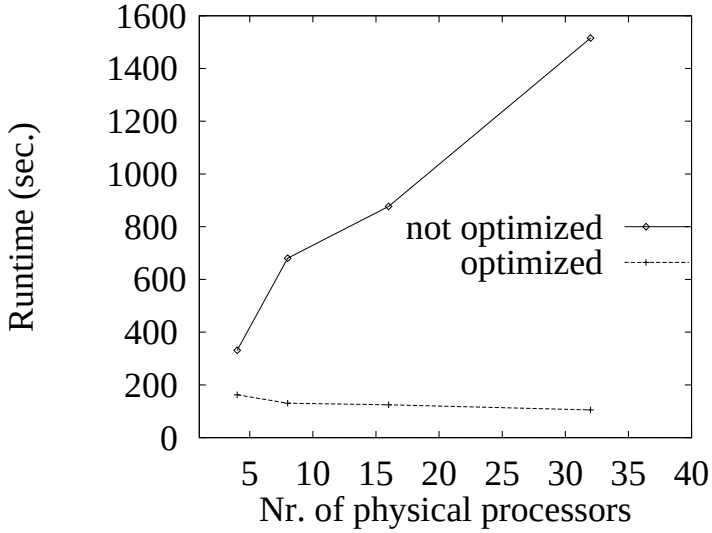

Figure 7. Runtime of the multigrid program: comparison of the non-optimized program, using one virtual process per processor, and the optimized program, where external messages are combined and communication is overlapped with communication.

performs a Monte–Carlo simulation of the particle transport, while TRACE uses a finite–element discretization of the model equations. To allow for a large number of elements — which are needed for realistic simulations — the program was parallelized at the Central Institute for Applied Mathematics (ZAM) in Jülich using a domain decomposition [32]. Following Schwarz' method [33], the linear equation that has to be solved in each timestep, is solved separately in each domain. Then the boundaries are exchanged between the domains and the process is repeated until a global convergence is reached.

In an earlier metacomputing project of the FZ Jülich and the GMD, 'Distributed Massively Parallel Computer' [23], the IBM SP2 of the GMD was coupled with the Intel Paragon XP/S 10 in Jülich via an OC–3 ATM line. For this metacomputer a message–passing library, that would allow applications to run distributed, was necessary. As at that time no such library existed, an especially adapted version (7.X) of the Parmacs library [34] was implemented by Pallas for the needs of the project. Parmacs made the heterogeneity of the metacomputer completely transparent for the application. Each computing node could communicate with every other node of the metacomputer regardless of its location. The obvious advantage of this approach is that existing message–passing applications like TRACE could easily be ported to the metacomputer.

The problem, however, is that the application is not aware of the heterogeneity of the metacomputer. Huge differences in bandwidth and latency between communication inside and between the two massively parallel computers result in dramatically increased communication time and reduced efficiency. A further complication is that the difference in floating–point performance leads to a load–imbalance although the program is perfectly balanced on a homogeneous system.

Efforts have been made to overcome these problems with some success. The load–

imbalance was resolved by choosing the domain sizes in the decomposition proportional to the floating–point performance of the computing nodes. The communication bottleneck was addressed by modifying the topology of the domain decomposition such that only one pair of nodes had to exchange messages over the external connection [22]. Nevertheless the main result of these investigations is that a tightly–coupled homogeneous application is not a good candidate for metacomputing due to the gap between internal and external communication performance.

Therefore, a different strategy is pursued in our current metacomputing activities. In their present versions, TRACE and PARTRACE are independent programs that 'communicate' via files. TRACE simulates the water flow until a stationary flow evolves and writes the resulting fields into a file which is then used as input for the particle simulation that is done by PARTRACE. The main reason for this splitting was that the memory requirements for a resonable spatial resolution and number of particles could not be fulfilled simultaneously.

It is considered a serious restriction of this approach that the simulation of particle transport is limited to stationary flows. Running both applications simultaneously on a metacomputer and exchanging the data via message–passing will resolve this limitation. This approach is well–suited for our metacomputing environment for the following reasons. The applications are only loosely coupled, in the sense that an exchange of data only takes place at most once per timestep. This involves a large amount of data, because the complete fields representing the flow have to be communicated between the two machines. Inside the program TRACE less data (only boundaries between the domains) have to be exchanged, but more often: about 10 to 15 times per timestep until global convergence of the Schwarz method is achieved. Communicating the fields representing the water flow from TRACE to PARTRACE still requires a high bandwidth — we expect up to 300 MBit/s for realistic simulations. The main advantage of the new approach is that, although the latency in the testbed is rather high, it is not a critical factor, since information flows in one direction only. A final point is that some kind of static load–balancing has to be introduced when running TRACE and PARTRACE on two computers. This can simply be done by choosing the number of processors used in each machine appropriately.

The basis of the coupled application is the 'homogeneous' version of TRACE, which does not include the earlier metacomputing–specific optimizations. This is due to the fact that TRACE itself will not be run distributed and the metacomputing-optimizations introduce some overhead into the program. However, running TRACE distributed as in the earlier experiment on the new metacomputer yields some rule–of–a–thumb factor of the improvements since the former project. It should be noted that a lot of things have changed since then, so this factor may vary from application to application.

- The Paragon was replaced by a T3E-900. Besides the higher performance of the T3E, this removed the additional overhead of data conversion. T3E and SP2 both use IEEE–arithmetic with the same byte order. Only the lenghts of the integer datatypes differ.

- The network connection is faster. In the former project, the gross bandwidth was 34 Mbit/s. The OC–3 ATM interface of the T3E is the limiting factor in our current configuration.

Table 2
Execution time of TRACE for different (meta–)computer configurations and communication libraries.

	Library	No. of Nodes	exec. time
SP2	Parmacs 7	4	226
		8	152
XPS	Parmacs 7	4	1106
		8	735
XPS — SP2	Parmacs 7	2+2	1173
		4+4	1398
SP2	MPI	4	147
		8	101
T3E	MPI	4	77
		8	47
T3E — SP2	MPI–PACX	2+2	190
		4+4	208

- The communication library was Parmacs then and now is MPI–PACX. PACX had to be modified to support the coupling of a T3E and an SP2. This was not difficult, because PACX already contained some mechanisms to support heterogeneity and the data types on both machines are almost identical.

- Some new features of TRACE have not been used in the comparison, so they should not have influenced execution time. Porting TRACE from Parmacs to MPI had some effect, as can be seen from the results.

The results of our measurements are shown in Table 2. All of them use a version of TRACE that does not contain any metacomputing–specific optimizations. The test case is a 31^3 grid. The execution times given are for 15 time–steps and do not include the time needed for reading the initial data from disk and writing the results back to disk.

As can be seen, the execution time on the SP2 alone has been reduced about 35%. This is due to the change of the communication library. While Parmacs was layered on top of the SP2–native MPL, we now use the native MPI implementation by IBM. Improvements in the compilers might also have some influence.

In both metacomputers, load–imbalance is introduced by the different performance of the components. As was demonstrated in detail by the former project, the effective floating–point performance of the metacomputer is determined by the slower machine [23], because the additional power of the faster machine is wasted by idle waiting for communication with the slower one. The same effect can of course be seen here. Whereas the execution time on the XPS/10 with Parmacs was more than 7 times longer than on the SP2 with MPI, the bandwidth and latency of the network connection have only been improved by a factor of 4. Therefore the scaling of the application — when run on the metacomputer — should be worse than before. This is exactly what we observe: the penalty for running TRACE distributed is larger for both the 2+2 and the 4+4 processors case. The fact that the slow–down from 2+2 to 4+4 processors is smaller is

due to optimizations in PACX that reduce the extra effort for sending smaller messages. This supports our conclusion that tightly coupled homogeneous applications should not be run on a metacomputer (at least without extra efforts like those described in section 3.2).

4.2. Current Activities, Future Plans

In the case of TRACE/PARTRACE the motivation for metacomputing is mainly the size of the problem. The heterogeneous structure of the problem qualifies it for our geographically distributed environment. Besides that, several other applications that can benefit from metacomputing for different reasons, are currently under investigation — in cooperation of several institutes in the FZ Jülich, the GMD and other locations — as part of the Gigabit Testbed West project.

Two applications arise from experiments on brain activity, which are performed by the Institute of Medicine in the FZ Jülich. One of them is the Analysis of magnetoezephalography data. The magnetic field around a human head is measured with an array of superconducting quantum interference devices (SQUIDs). From these data, the distribution of electric currents in the brain can be reconstructed by solving an inverse problem. In Jülich, this is done with the 'Multiple Signal Classification' (MUSIC) algorithm [35]. With MUSIC parameters of a finite number of current dipoles are obtained in two phases [36]. The positions of the dipoles are estimated in phase 1, orientation and strength in phase 2. Phase 1 involves a nonlinear optimization, which is done best on a massively parallel system. Phase 2 is a least–squares problem, which is well–suited for a vector computer. In the current configuration, execution time is not really critical, since the calculations don't have to be performed in realtime. Nevertheless, this application is a good candidate for heterogeneous metacomputing, because it can benefit from running distributed on different supercomputer architectures and does not suffer from latency problems, since data has to be transfered mainly in one direction — from phase 1 to phase 2.

Another experiment in Jülich that deals with brain activity is based on Magnetic Resonance (MR) Tomography. Here a test person is exposed to e.g. periodic visual or acoustic stimulations. The areas of brain activity are identified from temporal correlations of the MR data. Therefore time series of tomographic data have to be corrected for head movement and then correlated. The results are visualized in 3–D. In order to allow interactive response of the experimentalist, all this has to be done in realtime. The high hardware requirements for this project can be met by the project partners in the FZ Jülich and the GMD. The necessary computing power is available in both locations, a tomograph in Jülich and an SGI visualization server (as well as the visualization know–how) in the GMD. It should be noted that a similar application has recently been demonstrated by the Pittsburgh Supercomputing Center [37].

Another metacomputing project that will use the Jülich–St. Augustin metacomputer deals with the distributed calculation of climate and weather models. Here, the Alfred–Wegener–Institute (AWI), the German Climate Computing Center (DKRZ) and the GMD will use the supercomputers in Jülich and St. Augustin for a coupled simulation of atmospheric processes and the ocean–ice system. The approach is promising, because these subsystems are only loosely coupled — by the exchange of energy and water at the ocean

surface. For this purpose existing codes will be modified and coupled. Further projects like 'Multimedia applications in a Gigabit–WAN' have been set up in the framework of the Gigabit Testbed West, but are beyond the scope of this paper.

4.3. Transatlantic Metacomputing applications

In the frame of the metacomputing projects at RUS a transatlantic cooperation was set up to run applications on the T3Es of Pittsburgh Supercomputing Center (PSC) and the High Performance Computing Center at Stuttgart. Additionally the research group had access to a T3E at San Diego Supercomputing Center. So tests could be performed either across the transatlantic connection or via vBNS.

The critical point for an application in such a metacomputing scenario is the latency. Latencies across the Atlantic go up to about 70 milliseconds imposed by the hardware only. PACX–MPI adds some 4-5 milliseconds to that. The most critical point for an application is therefore to hide away as much as possible latency. Other points that inhibit performance are the following:

- Parallel I/O: A lot of applications still dedicate one node to read in the data at the beginning and distribute all information to the other nodes. While on an MPP the overhead caused may still be acceptable in a metacomputing scenario this is a serious bottleneck. This problem can be seen in our applications. Future work in metacomputing will have to deal with the problem not to have only parallel I/O on one single system - as will be provided by MPI-2 very soon - but to find also ways to distribute I/O across a cluster of MPPs.

- In consequence this will require to provide the user with a distributed file system that allows for handling of distributed data.

- Based on such a distributed file system another critical feature will be the distributed visualization of data. At HLRS a distributed visualization environment was developed that will provide such functionality [18].

4.3.1. URANUS

The Navier-Stokes solver URANUS (Upwind Relaxation Algorithm for Non-equilibrium flows of the University of Stuttgart) was developed at the Institute for Space Systems at the University of Stuttgart [19]. It is used for the simulation of non-equilibrium 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 [20]. 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 (pre-processing, processing and post-processing). Pre-processing is still done sequentially. One node reads in all data, does some pre-processing 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.

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 622 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.

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 the overall time that it takes to solve a medium sized problem (880.000 grid cells). Timings as given here include pre-processing, processing and post-processing. 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.

Running 10 iterations we compare the results of a simulation on a single machine using 128 nodes and on two machines using 2 times 64 nodes. A first test run shows that time for the processing part goes up from 102.4 seconds on one machine to 157 seconds on two machines. Obviously the overhead imposed by the higher latency is more than 50 percent. Time for pre-processing goes up from 272 seconds to 508 seconds. The code than was adapted for metacomputing a little bit. This meant to reduce the number of global communications. On one machine this reduced processing time to about 91 seconds and

pre-processing time to about 269 seconds. In the metacomputing environment processing time was reduced to 150 seconds and pre-processing time to about 480 seconds. In the next step asynchronous message-passing was introduced. It helped to reduce processing time to 117 seconds for the metacomputing scenario.

It is obvious that PACX–MPI imposes such a high overhead on the communication by using TCP that for the non-optimized version of PACX–MPI 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.

4.3.2. P3T-PSMC

P3T-PSMC is an object oriented Direct Simulation Monte Carlo Code that employs an orthogonal design based on the P3T (Parallel/Physics/Particle 3D Tools) kit in development at the Institute for Computer Applications of the Stuttgart University for general particle tracking applications.

Monte Carlo codes are well suited for metacomputing since they show an excellent ratio of communication and computation. First results in the metacomputing scenario on the transatlantic connection show good performance. For small number of particles the metacomputing shows some overhead. But already for about 15000 particles no time difference can be seen between running the code on 60 nodes of one machine or on 30 nodes each of two machines. Up to 125000 particles timings for one time step are the same. Only for 500000 particles a difference of about 3 percent can be seen. This indicates that the amount of data to be sent is too large to still be handled by the 2 Mbit/s connection. However, during Supercomputing 97 based on PACX-MPI the program was able to set a new world record for molecular dynamics simulating a crystal with 1.4 billion particles on two T3Es using 1024 processors.

5. CONCLUSION

The examples in this contribution show that various problems we have to face in metacomputing environments are quite well understood. The evolution of the Wide Area networks is continuously enhancing the available bandwidth, but the latency is already approaching the limit imposed by the speed of light. Therefore applications have to be selected carefully for metacomputing. To keep the effort for porting applications to a metacomputer acceptable supporting tools and libraries are essential. With PACX–MPI, an easy–to–use yet efficient message–passing library has been developed and is used successfully in several metacomputing applications. Experiments with several such applications show that latency hiding and load–balancing are important when performance is an issue. A library that supports these two key–points is currently under development and shows promising preliminary results.

Other concepts for optimization still have to be investigated. Using parallel I/O and making explicit use of the heterogeneity of the applications by appropriately mapping it on the different components of the metacomputer are such concepts. Also, using networking protocols different from TCP/IP can substantially reduce latency and enhance bandwidth

— when the distance between the connected supercomputers is not too large. Several projects which are just starting up address these points.

Acknowledgments

The authors gratefully acknowledge support from Pittsburgh Supercomputing Center and supercomputing time provided by the San Diego Supercomputing Center. We also wish to thank the BMBF for funding parts of this work and the DFN for its support.

REFERENCES

1. C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, R.V. Kenyon, and J.C. Hart, The CAVE: Audio Visual Experience Automatic Virtual Environment, *Communications of the ACM*, Vol. 35, No. 6, pp. 65-72, 1992.
2. D. Erwin, The UNICORE Architecture and Project Plan, *Workshop on Seamless Computing*, ECMWF, Reading, September 16–17, 1997.
3. V. Sander, High Performance Computer Management, *Workshop Hypercomputing*, Rostock, September 8–11, 1997.
4. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, University of Tennessee, http://www.mcs.anl.gov/mpi/index.html, 1995.
5. Roch Bourbonnais, The Thinking behind SUN's MPI Machines, *The Fourth EurPVM-MPI Users' Group Meeting*, Cracow, Poland, November 3-5, 1997.
6. Paco Romero, Message Passing Interface on HP Exemplar Systems, *The Fourth EurPVM-MPI Users' Group Meeting*, Cracow, Poland, November 3-5, 1997.
7. William Gropp, Ewing Lusk, Nathan Doss, Anthony Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, 22(6):789-828, September 1996.
8. A. Geist, PVM 3 User's Guide and Reference Manual, ORNL/TM-12187, 1994.
9. G.E. Fagg, J.J. Dongarra, PVMPI: An Integration of the PVM and MPI Systems, Department of Computer Science Technical Report CS-96-328, University of Tennessee, 1996.
10. M. Brune, J. Gehring and A. Reinefeld, A lightweight Communication Interface for Parallel Programming Environments, in *High-Performance Computing and Networking HPCN'97*, Springer, Berlin, 1997.
11. F-C. Cheng, P. Vaughan, D. Reese, A. Skjellum, The Unify System, Technical Report, NSF Engineering Research Center, Mississippi State University, 1994.
12. 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*, pages 91-98, Springer-Verlag Berlin Heidelberg, 1997.
13. T. Beisel, Ein effizientes Message-Passing-Interface (MPI) für HiPPI. Diplomarbeit, RUS, 1996. In German.
14. E. Gabriel, Erweiterung einer MPI-Umgebung zur Interoperabilität verteilter MPP-Systeme, Studienarbeit, RUS-37, 1997. In German.
15. T. Beisel, E. Gabriel, M. 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, pages 75-83, Springer-Verlag Berlin Heidelberg, 1997.
16. W.R. Stevens, *UNIX Network Programming*, Prentice Hall, Englewood Cliffs, 1990.
17. M.M. Resch, T. Beisel, T. Boenisch, B. Loftis, R. Reddy, Performance Issues of Intercontinental Computing, *Cray User Group Conference*, 1997.
18. A. Wierse, Performance of the COVISE visualization system under different conditions in Visual Data Exploration and Analysis II, in Georges G. Grinstein, Robert F. Erbacher eds., *Proc. SPIE 2410*, pages 218-229, San Jose, 1995.
19. H.-H. Fruehauf,O. Knab, A. Daiss, U. Gerlinger, The URANUS code - an advanced simulation tool for reentry nonequilibrum flow simulations, *Journal of Flight Sciences and Space Research*, 19 (1995) pp. 219-227.
20. T. Boenisch, R. Ruehle, Portable Parallelization of a 3-D Flow-Solver, in*Parallel Comp. Fluid Dynamics '97* (Elsevier, Amsterdam, 1997) to appear.
21. O. A. McBryan, HPCC: The Interrelationship of Computing and Communication, in: E. D. Hollander, G. R. Joubert, F. J. Peters, D. Trystran (editors): *PARALLEL COMPUTING: State-of-the-Art and Perspectives.* Elsevier Science B.V., 1996.
22. J. Henrichs, M. Weber, W. E. Nagel, R. Völpel, H. Grund, Metacomputing in a Regional ATM-Testbed - Experience with Reality -, *Proceedings of the ParCo'97 Conference*, North-Holland, Amsterdam, to appear.
23. J. Henrichs, W. E. Nagel, M. Weber, R. Völpel, H. Grund, Abschlußbericht des Teilprojektes "Höchstleistungsrechenzentrum: Verteilter massiv-paralleler Rechner" im "Regionalen Testbed Nordrhein-Westfalen (RTB-NRW)", Forschungszentrum Jülich, ZAM, Technical Report FZJ-ZAM-IB-9711, Juli 1997, in German.
24. H. Fukumori, Y. Kono, K. Nishimatsu, and Y. Muraoka, Finite Element Analysis with Heterogeneous Parallel Computer Environment over ATM Network, *Proceedings of I-SPAN'96*: International Symposium on Parallel Architectures, Algorithms, and Networks, 1996.
25. D. J. Morton and J. M. Tyler, An Integrated Scheme for the Distribution of Adaptive Finite Element Code in the Cray Y-MP/T3D Computing Environment, Technical Report arsc-sp-117-95, Arctic Region Supercomputing Center, University of Alaska, Fairbanks, 1995.
26. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, V. Sunderam, PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, Cambridge, MA., 1994.
27. W. D. Gropp, Parallel Computing and Domain Decomposition, Technical Report Mathematics and Computer Science Division, Argonne National Laboratory, MCS-P257-0891, 1991.
28. V. Kumar, A. Grama, A. Gupta, G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms.* The Benjamin/Cummings Publishing Company, Inc., 1994.
29. C. Perez, Load Balancing HPF programs by migrating virtual processors, Rapport de recherche de l'INRIA - Rhône-Alpes, RR-3037, 1996.
30. H. Vereecken, G. Lindenmayr, A. Kuhr, D.W. Welte, and A. Basermann, Numerical Modelling of Field Scale Transport in Heterogeneous Variably Saturated Porous Media, Forschungszentrum Jülich, ZAM, Technical Report IB-9301, 1993.

31. G.T. Yeh, 3DFEMWATER: A Three Dimensional Finite Element Model of Water Flow through Saturated–Unsaturated Media, Oak Ridge National Laboratory, Publication No. 2904, 1987.
32. R. Wimmershoff: Entwicklung und Implementierung einer dreidimensionalen Partitionierungsstrategie für das Programm TRACE auf einem massiv parallelen Rechner. Technical Report Forschungszentrum Jülich, Jül–3157, 1995, in German.
33. M. Dryja and O.B. Widlund, Multilevel Additive Methods for Elliptic Finite Element Problems, in *Parallel Algorithms for Partial Differential Equations,* W. Hackbusch, Ed., Proceedings of the Sixth GAMM–Seminar, F. Fieweg, Braunschweig, pp. 58–69, 1990.
34. R. Calkin et al., Portable programming with the PARMACS message–passing library, *Parallel Computing*, 20(4):615–632, April 1994.
35. J.C. Mosher, P.S. Lewis, and R.M. Leahy, Multiple Dipole Modeling and Localization from Spatio–Temporal MEG DATA. *IEEE Trans. Biomed. Eng. 39*, pp. 541–557, 1992.
36. R. Beucker and H.A. Schlitt, Objective Signal Subspace Determination for MEG, Forschungszentrum Jülich, ZAM, FZJ–ZAM–IB–9715, 1997.
37. N.H. Goddard, G. Hood, J.D. Cohen, W.F. Eddy, C.R. Genovese, D.C. Noll, and L.E. Nystrom, Online Analysis of Functional MRI Datasets on Parallel Platforms. *Journal of Supercomputing*, in press.

Katalog der wissenschaftlichen Publikationen des ZAM (Stand: 25.01.99)

Die mit *ftp* gekennzeichneten Publikationen stehen im PostScript-Format auf dem Anonymous ftp-Server des ZAM **(ftp.zam.kfa-juelich.de)** im Verzeichnis **pub/zamdoc/ib/ib-9x** oder **pub/zamdoc/juel** zur Verfügung.
Bitte richten Sie Bestellungen mit E-Mail an **literatur.zam@fz-juelich.de** oder an:

Zentralinstitut für Angewandte Mathematik
FORSCHUNGSZENTRUM JÜLICH GmbH
Informationszentrum
D-52425 Jülich

IB-9801 ***ftp*** Ulrike Begiebing, Volker Sander: *ServerVision*

IB-9802 ***ftp*** Werner Anrath, Rainer Grallert: *Aufbau von NT-basierten Arbeitsgruppen in TCP/IP-Netzen am Beispiel Forschungszentrum Jülich*

IB-9803 ***ftp*** Felix Wolf, Bernd Mohr: *EARL - A Programmable and Extensible Toolkit for Analyzing Event Traces of Message Passing Programs -*

IB-9804 ***ftp*** Friedel Hoßfeld: *Verbund der Supercomputer-Zentren in Deutschland - Ansichten, Einsichten, Aussichten*

IB-9805 ***ftp*** Jörg Henrichs: *Optimizing and Load Balancing Metacomputing Applications*

IB-9806 ***ftp*** Volker Sander, Dietmar Erwin, Valentina Huber: *High-Performance Computer Management Based on Java*

IB-9807 ***ftp*** Jürgen Meißburger: *„zammon" - Ein Webserver für das JuNet-Management*

IB-9808 ***ftp*** Michael Gerndt, Bernd Mohr, Mario Pantano, Felix Wolf: *Automatic Performance Analysis for CRAY T3E*

IB-9809 ***ftp*** Jim Galarowicz, Bernd Mohr: *Analyzing Message Passing Programs on the Cray T3E with PAT and VAMPIR*

IB-9810 Rudolf Berrendorf: *Optimizing Load Balance and Communication on Parallel Computers with Distributed Shared Memory*

IB-9811 ***ftp*** Bart Theelen: *Wrappers for Tracing Collective Communication Functions with PAT*

IB-9812 ***ftp*** Johannes Grotendorst, Jürgen Dornseiffer: *Computer-aided Modelling and Simulation of the Thermodynamics of Steam Reforming*

IB-9813 ***ftp*** Volker Sander, Lothar Wollschläger: *RAID-Systeme: Durchsatz im Überfluß?*

IB-9814 ***ftp*** Ralph Niederberger, Leonhard Radermacher, Karl Milz: *Bits für Kids - Modellversuch zur Unterstützung Jülicher Schulen beim Internetzugang*

IB-9816 ***ftp*** Rudolf Berrendorf, Heinz Ziegler: *PCL - The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors*

IB-9819 ***ftp*** Marlene Busch, Heinz Heer, Michael Wagener: *Praxisbezogene Einführung in IDL*

IB-9820 Johannes Grotendorst: *Mathematik mit Maple - Eine Einführung mit Beispielen aus der Analysis und Linearen Algebra*

IB-9821 ***ftp*** Christian Bischof, Friedel Hoßfeld: *Technisch-wissenschaftliches Hochleistungsrechnen: Herausforderungen komplexer Systeme an die Computer-Simulation*

IB-9824 Thomas Eickermann, Jörg Henrichs, Michael Resch, Robert Stoy, Roland Völpel: *Metacomputing in Gigabit Environments: Networks, Tools, and Applications*

IB-9901 ***ftp*** Friedel Hoßfeld: *Teraflops Computing: A Challenge to Parallel Numerics*

Jül-3551 ***ftp*** Felix Wolf: *EARL - Eine programmierbare Umgebung zur Bewertung paralleler Prozesse auf Message-Passing-Systemen*

Jül-3552 ***ftp*** Rudolf Berrendorf: *Benutzer- und datengesteuertes Schleifen-Scheduling auf Parallelrechnern mit Distributed Shared Memory*

Jül-3581 ***ftp*** Volker Lempert: *Methoden und Untersuchungen zur Optimierung des Datendurchsatzes in ATM-Netzen*

Jül-3587 G. Egerer, R. Knecht, W.E. Nagel: *Algorithmen und Strukturen in C*

- Rüdiger Esser; Johannes Grotendorst; Marius Lewerenz (eds.): *Höchstleistungsrechnen in der Chemie*
- A. Kraus, O. Selke, F. Wissmann, J. Ahrens, H. -J. Arends, R. Beck, G. Galler, M. -Th. Hütt, B. Körfgen, J. Peise, M. Schumacher, F. Smend, R. Wichmann: *Angular and polarization dependence of Compton scattering from ^{4}He in the Δ-resonance region*
- Ralf Wilhelm, Olaf Heller, Manuela Bohland, Cordula Tomaschewski, Inge Klein, Peter Klauth, Wolfgang Tappe, Joost Groeneweg, Carl Johannes Soeder, Paul Jansen, Wolfgang Meyer: *Biometric analysis of physiologically structured pure bacterial cultures recovering from starvation*
- G. Widman, K. Lehnertz, P. Jansen, W. Meyer, W. Burr, C.E. Elger: *A fast general purpose algorithm for the computation of auto- and cross-correlation integrals from single channel data*