# Optimising MPI Applications for Heterogeneous Coupled Clusters with MetaMPICH

Carsten Clauss, Martin Pöppe, Thomas Bemmerl
Lehrstuhl für Betriebssysteme
RWTH Aachen University
Kopernikusstr. 16, 52056 Aachen, Germany
E-mail: {carsten, martin, thomas}@lfbs.rwth-aachen.de

## Abstract

*Cluster systems built mainly from commodity hardware components have become more and more usable for high performance computing tasks in the past few years. To increase the parallelism for applications, it is often desirable to combine those clusters to a higher lever, commonly called metacomputer. This class of high performance computing platforms can be understood as a cluster of clusters, where each cluster provides different processors, memory performance, cluster interconnect and external networking facilities. Our project called MetaMPICH provides a transparent MPI-1 implementation for those inhomogeneous cluster systems. While this feature of transparency makes porting existing MPI applications to metacomputers quite simple, the slowest network connection and the slowest processor will dominate the performance and scalability due to the lack of opacity. This paper describes our approaches to adapt iterative, grid based simulation algorithms to the structures of such heterogeneous coupled clusters.*

## 1. Introduction

With the emergence of cluster systems built from commodity hardware components in the field of high performance computing, middleware and application developers were confronted with new challenges in software development. The new systems were not as easy to use as dedicated parallel computers, because the components of the cluster systems were originally designed for desktop purposes. Many efforts have been made in the past years to solve these problems, adding communication libraries and services as well as process and resource management for distributed and parallel computing to the operating systems of the cluster nodes. Our own project called MetaMPICH [5] provides a MPI-1 [7] implementation for heterogeneous clustered systems. These systems can be understood as *clusters of clusters*, where each cluster exhibits different capabilities regarding processors, memory performance, cluster interconnect and external networking facilities.

The first development goal of MetaMPICH was to provide a transparent MPI-system for the applications in the Gigabit Testbed West project [2], in which an IBM SP2 was coupled with a Cray T3E Supercomputer via a 2,4 Giga-Bit ATM connection.

As ongoing work, we developed support for the nowadays widely used NORMA (NO Remote Memory Access) and NUMA (Non Uniform Memory Access) cluster architectures. While the MetaMPICH library still provides a transparent MPI-system, an additional MPI-communicator `MPI_COMM_LOCAL` reflects the group of processors on the local parallel system, which we call a *metahost* [4]. This local group uses a fast network to exchange MPI-messages and is a homogeneous cluster or a SMP-system, which implies that communication and computation is very efficient in this local group, because there is no data conversion, load balancing or network overhead necessary.

Figure 1 shows an example for a metacomputer configuration, in which the processes are grouped into three levels: metahost `A` is a cluster of dual-processor systems, representing the lowest grouping level 1, while metahost `B` is a SMP system whose processors are grouped in the higher level 2. As the sets of all processors in each metahost represent the second level (`MPI_COMM_LOCAL`), the third level constitutes the metacomputer itself in the MPI communicator (`MPI_COMM_WORLD`), which is the communicator containing all available MPI processors.

## 2. Adapting Applications to the Structures of Metacomputers

MetaMPICH's feature of transparency hides the inhomogeneous communication structures of the metacomputer

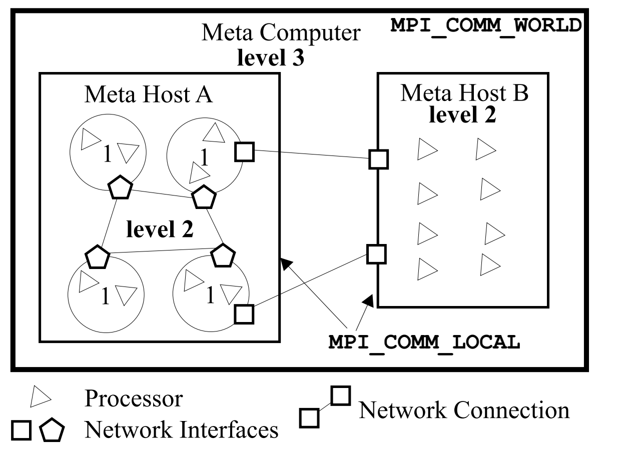

**Figure 1. Process Groups in a Meta Computer**

from the application. While this simplifies the porting of existing MPI applications to metacomputers, the slowest network connection and the slowest processor will dominate the performance and scalability due to the lack of opacity. Although the new communicator MPI_COMM_LOCAL allows to differentiate explicitly between inner-cluster nodes and nodes on other metahosts, its use demands a redesign of the application's communication routines – and in each redesign the knowledge of the respective metacomputer's characteristics needs to be taken into account.

Therefore, it would be desirable to create a new intermediate layer on top of MetaMPICH, which a number of applications can easily attach to and which simultaneously respects the particularities of the metacomputer. This approach allows to benefit from a transparent view for the applications and from the ability to provide load balancing (depending on performance characteristics) and to avoid bottlenecks in communication.

How to create such a new layer and which possibilities exist to decrease those bottlenecks is shown below for the important class of iterative, grid based simulation algorithms. This class of grid based algorithms contains all those whose cores solve discrete boundary value problems by iterative relaxation methods. The best known versions are the Jacobi and the Gauss-Seidel method [6].

## 3. An Approach to Optimise Grid Based Algorithms for Metacomputers

In parallel systems, a fair distribution of computational load is mandatory for efficient use of the computing resources. Therefore, various strategies of load balancing for different kinds of parallel systems have been developed. Applications running on a metacomputer require such a strategy as well, but due to the inhomogeneous structures it turns out to be much more difficult than on homogeneous systems [3][1][9]. Load balancing at runtime is often infeasible, because in this case the adjustment has to be done across the inter-metahost bottleneck. Instead of that, a fair a-priori load distribution among the metahosts by means of performance measurements is more preferable.

Another interesting approach is not to distribute the load explicitly, but to balance the computational power by introducing so-called *virtual processors*, as shown in [8]. In this model, stronger physical processors have to manage more virtual processors than weaker ones. The disadvantage of this concept is that scheduling those virtual processors and the additional communication among them will cause a non-negligible overhead.

Besides a fair load distribution, communication is a key factor. In grid based algorithms, each process works on a section of the entire grid, so that values on borders of those sections have to be exchanged between the processes. Thereby, the effort of communication for a process depends on the border length of these sections and on the location of its neighbours, because communication with neighbours in other metahosts will reduce performance.

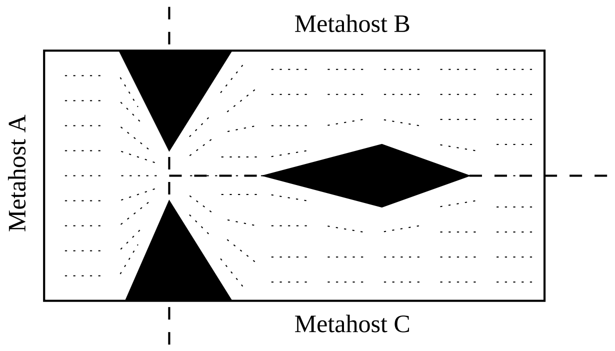

**Figure 2. Example: Flow Channel**

The question is whether the effort for inter-metahost communication can be reduced by partitioning these sections in a smart way. In most cases, the given boundary value problems specify additional boundary values within the grid. These inner boundary values are fixed or can be calculated by each process on its own. The underlying idea is now to disregard such fixed values during the inter-metahost communication phase, because they do not need to be exchanged. Hence, in case of a smart partition scheme, the message length and thus the communication effort between the metahosts will decrease. This concept is depicted in figure 2 for the application of a flow channel simulation.

## 4. An Implementation of an Adaptation Layer on MetaMPICH

Within the scope of our research, the mentioned approaches to adapt grid based algorithms to the structures of metacomputers were evaluated and finally implemented in form of a small programming interface between MetaMPICH and the applications. This interface called `MetaComm` provides simple communication functions, which can replace all explicit MPI function calls within a parallel application and can easily be included into serial applications to make them parallel. The functions are designed to be called by all application processes, whereby the target paradigm of SPMD (Single-Program-Multiple-Data) is strongly maintained.

The optimizing features of `MetaComm` are based on an appropriate predetermined partition scheme for the entire grid. During initialization, each single application process acquires its own grid section, including all inner boundary values. Afterwards, each process only needs to take care of the computational progress on its own section data. All required exchanges of values on borders of these sections are transparently performed by calling the new communication functions. In the first instance, `MetaComm` provides only two functions: `InitCommunication()` and `DoCommunication()`.

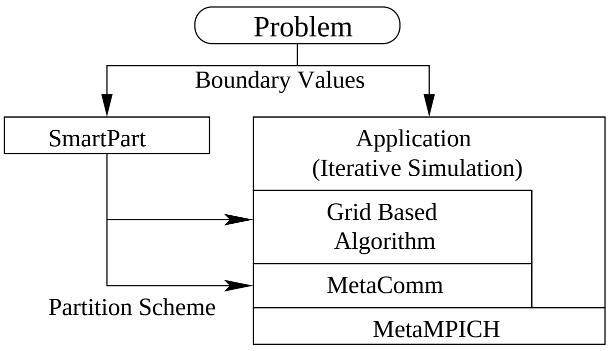

**Figure 3. Adaptation Layer on Top of MetaMPICH**

### 4.1. The Initialisation Phase

During the initialisation of `MetaComm` the composition of the respective metacomputer is analysed. This is basically done only by using the new MPI communicator `MPI_COMM_LOCAL`. Every process whose local rank is equal to zero sends the information about its local metahost size to the process with global rank zero. This process, playing a special role during the start-up phase, receives all those size messages until the sum is equal to the total number of processes. By analysing the correlation between the senders' global ranks and the information about the respective metahost sizes, the metacomputer's process arrangement can now easily be determined.

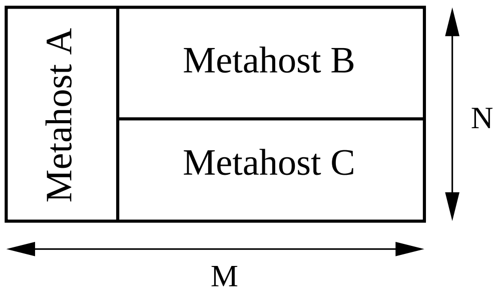

**Figure 4. Area Distribution onto the Metahosts**

Communication between metahosts will mainly be performed via the first and the last processes of the local rank lists, so in the following those processes are called the *gateways* of a metahost. The reason for this feature is that we want to centralise the message preparation for the inter-metahost communication within each metahost.

### 4.2. Distribution of the Partition Scheme

Assuming that all metahosts are working on an $N \times M$ grid, every metahost $x$ acquires its own $n_x \times m_x$ subsection, as it is predetermined in the partition scheme. Thereby, the partition scheme is explicitly passed to `MetaComm` via a special partition file, as shown as an example in figure 5.

```
N=160, M=240

MetahostA: n=160, m=80, OffsetN=0, OffsetM=0, horizontal
MetahostB: n=80, m=160, OffsetN=0, OffsetM=80, horizontal
MetahostC: n=80, m=160, OffsetN=80, OffsetM=80, vertical
```

**Figure 5. Example of a Partition File**

The further distribution of the metahosts subsections onto the individual processes is done in a simple algorithmic way, assuming an inner homogeneity of the metahosts' computational power. This redistribution can be done in a horizontal or a vertical manner. That means that the section of a metahost is divided by means of line or column distribution onto the processes, as can be seen in figure 6. Which kind of redistribution is to be chosen for a particular problem is also an issue of a smart partition scheme.

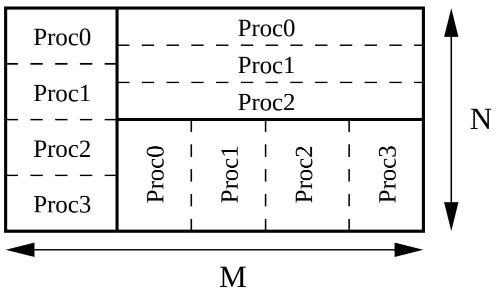

**Figure 6. Redistribution of the Subsections**

## 4.3. The Inner Boundary Values

To consider the inner boundary values in computation and communication, `MetaComm` manages an additional grid array where attributes of those boundary grid cells can be stored. This array can be filled by an explicit pattern, by algebraical functions or by specified polygons for an easy areal description.

The specifications of size and position of the local working areas are passed to the individual processes by the initialisation function. However, in most cases the processes do not need to know about their position within the entire grid, because all boundary values are transparently passed to them either by the communication functions or they can be considered by an additional probe function of `MetaComm`. This feature offers a total transparent view for the applications, in which no distinction between processes or metahosts needs to be taken into account.

## 4.4. The Communication Phase

By analysing the partition scheme in conjunction with the knowledge of the inner boundary values, the gateways of each metahost determine which cells must be exchanged and store this information in a so-called partner list.

When finishing a computation cycle, the parallelised application generally has to perform an exchange of border values among the processes. When using `MetaComm`, this is simply done by calling the `DoCommunication()` function. Apart from the grid values themselves, no further parameters need to be passed to this function, since all needed data structures have already been built during the initialisation phase.

**4.4.1. Communication across the Gateways** While the metahost-internal communication is performed in a common way, in case of an inter-metahost communication the gateways exchange the required values according to the appropriate partner lists. Thereby, one gateway always communicates with the top and the left adjacent neighbours, while the other gateway manages the communication with the bottom and the right partners. To avoid chronological order dependencies, firstly all inter-metahost messages are sent in a non-blocking mode, before the receiving of inter-metahost messages starts. As mentioned, only the inevitable values are exchanged.

**4.4.2. Left and Right Sided Communication** Assuming that the processes of a given metahost are distributed in a horizontal manner, the gateways initially have no access to all required values for left or right sided communication. Thus, the local process must firstly send those values to their respective gateways, which gather these messages and prepare them for the inter-metahost transmission. This procedure has the advantage that the inter-metahost communication is centralised, so that complex transmissions across this bottleneck are avoided. However, the disadvantage is an increased amount of interior communication, which is unfortunately proportional to the number of the local metahost processes. But, due to the assumed discrepancy between the interior and the inter-metahost communication, this proceeding seems to be an appropriate approach.

## 5. The Search for a Smart Partition Scheme

To find a smart partition scheme, which can then be used by `MetaComm`, we developed a software tool named `SmartPart`, which automates the search for the optimal partition for a given problem. By analysing the metacomputer's structure (number of metahosts, number of nodes in each metahost, etc.) and measuring the respective computational power, the tool can check every possible grid allocation, regarding the knowledge (bandwidth and latency) of the communication bottlenecks. Since `MetaComm` provides an almost arbitrary allocation of the grid and `SmartPart` iterates over all possibilities, an effective and optimal partition will in most cases be found. It may be emphasised that an *optimal partition* is always associated with a respective problem on one certain metacomputer.

## 5.1. Metrics and Scaling Factors

When searching for an optimal partition scheme, `SmartPart` is geared to so called scaling factors. These factors describe the metahosts by their ratio of computational effort (*time to calculate $n$ grid cells*) to communicational effort (*time to send $n$ grid cells to another metahost*). Those ratios can easily be determined by means of measurements in the applications or by using simple synthetic benchmarks.

In the current version of `SmartPart` it is assumed that the parameters for inter-metahost communication are the same for all participating metahosts. That means that exchanging a message between two metahosts A and B takes

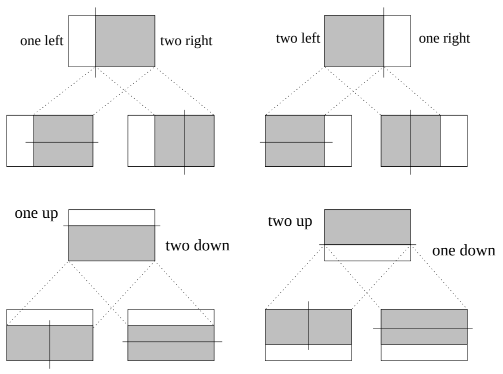

**Figure 7. Possible Division Patterns**

the same time as exchanging the same message between A or B and another metahost C. This assumption is necessary to figure out the relations between the computational powers.

To perform the grid division, the algorithm uses two recursive functions, each dividing a given (sub-) area into two parts. Thereby, the first function tries to divide the area by an optimal horizontal cut, while the second function divides the area in a vertical manner. Both functions are called by passing the dimension of the sub-area, the remaining number of metahosts to be assigned to this area and the respective scaling factors to them. To be able to decide which of those two functions delivers the better division policy for the current area, the functions return a metric, which is a rating for this actual cut (a small metric is equal to a good cut).

After each executed cut, both functions are applied to the two resulting sub-areas to analyse their further division, if there are still metahosts remaining to be assigned. This proceeding leads to the data structure of a weighted tree, wherein the best partition scheme can be recognised as the lightest path of metrics from top to bottom. In figure 7 you can see as an example the possible division patterns when dealing with three metahosts.

Since `SmartPart` iterates over all possible allocation patterns, this search is a NP-complete problem, but since the number of metahosts is probably less than ten it will only take a negligible amount of time.

### 5.2. An Analysis of Performance and Effects

In some cases of an optimal partition scheme, some metahosts get no section of the grid. This is the case when the bottleneck is so serious that including those metahosts means not to benefit from their computational power, but to have a dominant communication overload. However, with this kind of problems, the computational load increases quadratically with the problem size, while the communication load will only rise in a proportional way. Thus the allocation of those metahosts may recur at major problem sizes, where metacomputing will be worthwhile again.

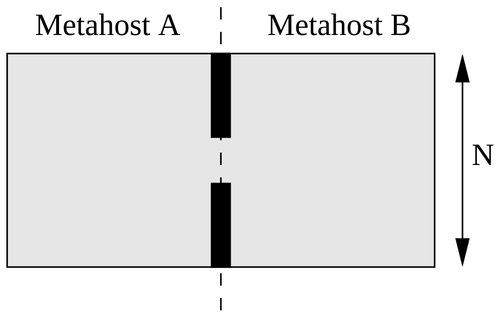

**Figure 8. A Simple Partition Example**

An example for a simple partition scheme is shown in figure 8, where two metahosts of equal computational power are working on a $N \times 2N$ grid divided by a focus barrier as can be found e.g. in a CFD simulation of a flow channel. In figure 9, the achieved reduction of communication time (measured on our own clusters) is plotted over the ratio of barrier length to grid height, and thus over the potential reduction of the message length.

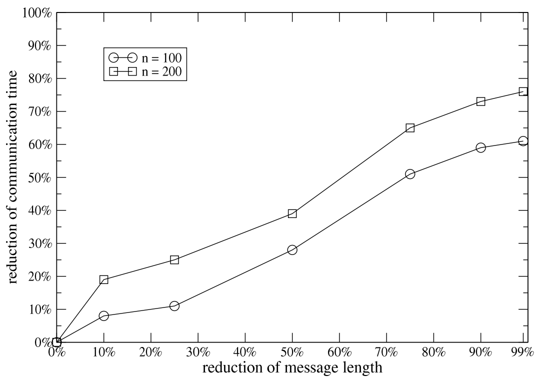

**Figure 9. Reduction of Communication Time**

As you can see, the resulting communication time does not vanish, even for a reduction up to 99% of the original message size. This is due to the inherent communication start up latency – but its influence will decrease for major grid sizes.

However, an increased number of grid cells also leads to a rise of the computational effort. In figure 10 you can see both the communication time for a message of $n$ cells and the computation time for $n \times n$ grid cells of a simple Poisson-problem. The breakeven points represent those grid sizes, where the computation takes more time than the inter-metahost communication. And for the considered example these are the grid sizes where the metacomputing begins to be worthwhile.

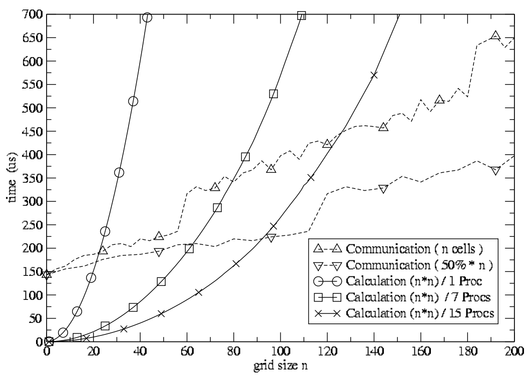

**Figure 10. Communication VS Computaion**

## 6. Conclusions and Outlook

The concept of metacomputing allows to bundle the distributed computational power and to dedicate it to a single problem. However, heterogeneous structures lead to a domination of bottlenecks on the achievable performance, so that the benefit of metacomputing should not be taken for granted. Nevertheless, the use of all available processors within a metacomputer is desirable for grid based algorithms which are not communication intensive.

The results of our research confirm that the approaches mentioned above are suitable for optimising this class of algorithms for metacomputers. `SmartPart` and `MetaComm` are powerful tools to improve the usability of the described metacomputing platforms and regain a transparent interface for an important class of parallel algorithms.

With the increasing acceptance of the use of computational grid infrastructures, the demand for larger parallelism within SPMD-applications will grow in the future. Thus, the integration of our tools in grid middleware environments will be a future scope of our work.

## References

[1] J. Brooke, S. Pickles, F. Costen, and S. Ord. Using metacomputing to process scientifi c data. Technical report, 1999.

[2] Eickermann, V¨olpel, Wunderling. Gigabit Testbed West Abschlussbericht. Technical report, Forschungszentrum J¨ulich, March 2000.

[3] J. Henrichs. Optimizing and Load Balancing Metacomputing Applications. Technical Report FZJ-ZAM-IB-9805, ZAM, Research Centre Juelich, Germany, 1998.

[4] M. P¨oppe and J. Worringen. *Meta-MPICH, user documentation & technical notes*. Lehrstuhl f¨ur Betriebssysteme, RWTH Aachen, 2002.

[5] Martin P¨oppe and Silke Schuch and Thomas Bemmerl. A Message Passing Interface Library for Inhomogeneous Coupled Clusters. In *International Parallel and Distributed Processing Symposium (IPDPS 2003), Workshop for Communication Architecture in Clusters (CAC 03)*, Nice, France, April 2003.

[6] T. Meis and U. Marcowitz. *Numerical solution of partial differential equations*. Springer Verlag, 1981.

[7] MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputing Applications*, 1994.

[8] C. Perez. Load balancing hpf programs by migrating virtual processors. Technical Report RR-3037, Unite de recherche INRIA Rhone-Alpes, 1996.

[9] S. Pickles, F. Costen, J. Brooke, E. Gabriel, M. Muller, M. Resch, and S. Ord. The problems and the solutions of the metacomputing experiment in sc99. In *HPCN Europe*, pages 22–31, 2000.