INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

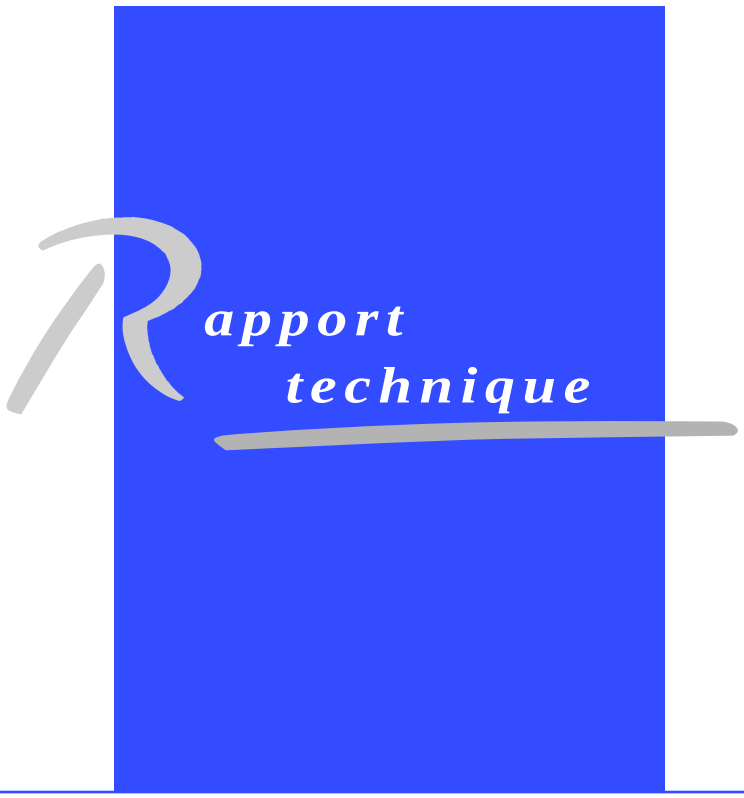
Improving MPI support for applications on hierarchically distributed resources

Stéphane Lanteri — Raúl López — Christian Pérez

N° 0336

May 2007

Thème NUM



ISSN 0249-0803 ISRN INRIA/RT--0336--FR+ENG

Improving MPI support for applications on hierarchically distributed resources

Stéphane Lanteri , Raúl López , Christian Pérez

Thème NUM —Systèmes num´eriques
Projets Paris, Caiman

Rapport technique n° 0336 —May 2007 — 10 pages

Abstract: Programming non-embarrassingly parallel scientific computing applications such as those involving the numerical resolution of system of PDEs using mesh based methods for grid computing environments is a complex and important issue. This work contributes to this goal by proposing some MPI extensions to let programmers deal with the hierarchical nature of the grid infrastructure thanks to a tree representation of the processes as well as the corresponding extension of collective and point-to-point operations. It leads in particular to support $N \times M$ communications with transparent data redistribution.

Key-words: MPI, Grids, tree structure, hierarchical communication, data redistribution.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 —Télécopie : +33 2 99 84 71 71

Une extension MPI pour les ressources distribues hiérarchiques

Résumé : La programmation des applications massivement parall´eles pour le calcul scientifique sur grilles est une probl´ematique complexe et importante. Un exemple de ce type d'applications est la r´esolution num´erique de systèmes à ´equations diff´erentielles partielles qui utilisent des m´ethodes de maillage. Le travail pr´esent´e dans ce rapport se propose d'´etendre MPI afin de permettre aux d´eveloppeurs d'avoir une approche de programmation parallèle guid´ee par la nature hi´erarchique des infrastructures de grilles. L'approche propos´ee est bas´ee sur une repr´esentation des processus en arbre et une extension des communications point à point et collectives correspondantes. En particulier, le travail se base sur la mise en place de communications $N \times M$ comprenant la redistribution transparente des donn´ees.

Mots-clés : MPI, Grilles, structure en arbre, communication hierarchique, redistribution de donn´ees.

1 Introduction and Motivation

Grid computing is currently the subject of a lot of research activities worldwide. It is particularly well suited to compute intensive, embarrassingly parallel scientific computing applications. The situation is less clear for non-embarrassingly parallel scientific computing applications such as those involving the numerical resolution of systems of PDEs (partial differential equations) using mesh based (finite difference, finite volume or finite element) methods. In most cases, grid-enabled numerical simulation tools are essentially a direct porting of parallel software previously developed for clusters of PCs or parallel supercomputers, thanks to the availability of appropriate MPI implementations such as MPICH-G2[1]. However, these grid-enabled simulation software rarely take into account important architectural issues characterizing computational grids, such as heterogeneity both of processing nodes and interconnection networks, which have a great impact on performance. Moreover, they are quite difficult to program. Considering that a computational grid can be seen as a hierarchical architecture, the general objective of the present work is to improve the support of MPI based scientific computing applications on hierarchically distributed resources. This work is taking place in the context of the DiscoGrid project[1]. Combining contributions of algorithmic nature with the adoption of modern programming principles and methodologies, the DiscoGrid project aims at demonstrating that a computational grid can be used as a high performance computing platform for non-embarrassingly parallel scientific computing applications.

The remainder of this report is divided as follows. Section 2 introduces a class of motivating application and Section 3 presents the related works. The proposed MPI extensions are dealt with in Section 4 while Section 5 concludes the report.

2 Motivating application and infrastructure

2.1 Motivating application

The targeted scientific computing applications take the form of three-dimensional finite element software for the simulation of electromagnetic wave propagation (computational electromagnetism - CEM) and compressible fluid flow (computational fluid dynamics - CFD). Such applications have largely and are still benefiting from parallel processing capabilities. The traditional way of designing scalable parallel software for the numerical resolution of systems of PDEs is to adopt a Single Program Multiple Data (SPMD) parallel programming model that combines an appropriate partitioning of the computational mesh (or the algebraic systems of equations resulting from the time and space discretization of the systems of PDEs defining the problem) and a message passing programming model while portability is maximized through the use of the Message Passing Interface (MPI). It is clear that such an approach does not directly transpose to computational grids despite the fact that appropriate, interoperable, implementations of MPI have been developed (e.g. MPICH-G2[1] in the Globus Toolkit[2]). By this, we mean that computational grids raise a number of issues that are rarely faced with when porting a scientific computing application relying on programming models

[1] Project number ANR-05-CIGC-005 funded by the French ANR under the framework of the 2005 program *Calcul Intensif et Grilles de Calcul*

and numerical algorithms devised for classical parallel systems. The most important of these issues with regards to parallel performances is heterogeneity (both of processing nodes and interconnection networks which impacts computation and communication performances). This heterogeneity issue is particularly challenging for unstructured mesh based CEM and CFD solvers because of two major issues. First, they involve iterative solution methods and time stepping schemes that are mostly synchronous thus requiring high performance networks to achieve scalability for a large number of processing nodes. Second, most if not all of the algorithms used for the partitioning of computational meshes (or algebraic systems of equations) do not take into account the variation of the characteristics of processing nodes and interconnection networks.

2.2 Infrastructure

A grid computing platform such as the Grid'5000 experimental test-bed can be viewed as a (at least) three level architecture: the highest level consists of a small number (< 10) of clusters with between a few hundreds and some thousand nodes. These clusters are interconnected by a wide area network (WAN) which, in the case of the Grid'5000 experimental test-bed, takes the form of dedicated 10 Gb/s links provided by Renater. The intermediate level is materialized by the system area network (SAN) connecting the nodes of a given cluster. This level is characterized by the fact that the SAN may differ from one cluster to the other (Gigabit Ethernet, Myrinet, Infiniband). The lowest level is used to represent the architecture of a node, single versus multiple-processor systems, single versus multiple-core systems and memory structure (SMP, Hypertransport, etc.).

2.3 Discussion

In the context of mesh based numerical methods for systems of PDEs, one possible alternative to the standard parallelization strategy that aims at dealing with the heterogeneity issue, consists in adopting a multi-level partitioning of the computational mesh. In this case, we can for instance assume that the higher level partitioning is guided by the total number of clusters while the lower level partitioning is performed locally (and concurrently) on each cluster based on the local numbers of processing nodes. Considering for illustration purpose the case of two-level partitioning, two types of artificial interfaces are defined: intra-level interfaces refer to the intersection between neighboring subdomains which belong to the same first-level partition while inter-level interfaces denote interfaces between two neighboring subdomains which belong to different first-level subdomains. Then, the parallelization can proceed along two strategies:

- a flat (or single-level) parallelization essentially exploiting the lower level partitioning and where the intra- and inter-level interfaces are treated in the same way. A grid-enabled MPI version such as MPICH-G2 is best suited to implementation of this strategy;
- a hierarchical approach where a distinction is made in the treatment of intra- and inter-level interfaces. In this case, we can proceed in two steps: (1) the exchange of information takes place between neighboring subdomains and in the higher level partitioning through inter-level interfaces, which involves a $N \times M$ redistribution operation; (2) within each higher level

subdomain, the exchange of information takes place between neighboring subdomains in the lower level partitioning through the intra-level interfaces.

3 Related Works

As far as we know, the related works can be divided in two categories. First, several efforts such as MPICH-G2 [1], OpenMPI [3], or GridMPI [4], have been done to provide efficient implementations of the flat MPI model on hierarchical infrastructures. The idea is to make use of hierarchy-aware algorithms and of a careful utilization of the networks for implementing collective communications.

In these works, the MPI runtime is aware of the hierarchy of resources but not the programmers. As this is a limiting factor for hierarchical applications like those presented in Section 2, there has been a proposal to extend the MPI model: MPICH-G2 provides an extended MPI with two attributes associated with every communicator to let the application discover the actual topology: `MPICHX_TOPOLOGY_DEPTHS` and `MPICHX_TOPOLOGY_COLORS`. The depths define the available levels of communication for each process, noting that MPICH-G2 defines 4 levels (WAN-TCP, LAN-TCP, intra-machine TCP, and native MPI). The colors is a mechanism that determines whether two processes can communicate at a given level: they can do so if and only if they have the same color for that level. While such a mechanism helps the programmers in managing the resource topology, it suffers several limitations: there is a hard coded number of levels so that some situations cannot be taken into account, like NUMA node[2]. Moreover, a communication infrastructure is associated with each level while application developers care more about network performance parameters like latency or bandwidth.

A second set of works deals with $N \times M$ communications such as InterComm [5] or PaCO++ [6]. Their goal is to relieve programmers from complex, error prone and resource dependent operations that involve data redistribution among processes. These works study how to easily express data distributions for programmers so as to automatically compute an efficient communication schedule with respect to the underlying network properties. They appear to be a complement to MPI, as they take into account communications from parallel subroutine to parallel subroutine, that is to say that they can be observed as the parallel extension of MPI point to point communications. Note that there are not yet standard APIs to describe data distributions, though there have been some standardization efforts to provide a common interface like the Data Reorganization Interface [7].

4 Improving MPI support for hierarchies of resources

Relying on hierarchically organized resources has become a major goal in the message passing paradigm as it has been exposed above. However, support given to users by existing frameworks does not always match their needs, as long as program control, data-flow and infrastructure's attributes are independently handled. We have identified some features that can be improved in this context, which are dealt with in the next sections.

[2]Communication performance within a NUMA node is not the same as through a network

4.1 Basic support

MPI modifications proposed here intend to bring together the advantages from the message passing paradigm and the hierarchical view of a computing system by the programmer. Those modifications are based on the utilization of a simple tree model, where processes are the leafs of the tree while nodes are just an abstraction of groups of processes (organized according to their topology). More precisely this tree can be observed as the hierarchical organization of resources at run-time. Figure 1 shows an example of such structure together with the hierarchically partitioned data used by an application.

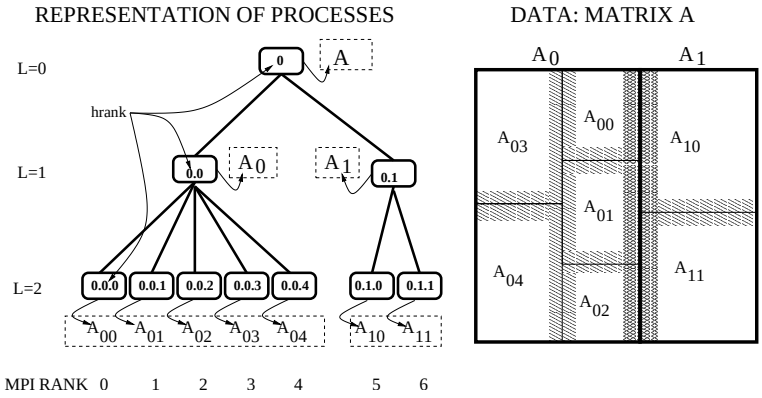


Figure 1: *Left:* Representation of processes derived from the structure of resources, last row indicates the rank in a standard communicator. *Right:* Data hierarchical partitioning. Dashed rectangles on the left show the association of data to resources.

Hierarchical identification system. We propose to introduce a new definition of rank, called `hrank` to denote its hierarchical nature. It consists of an array of identifiers that represents the tree nodes in the different levels to which the process belongs from the root to the bottom of the tree. This representation intrinsically involves the tree organization and easily identifies the resource grouping. In Figure 1 each node of the tree has been associated with its hrank.

Navigation of the tree structure The programmer will be given a set of functions to explore the tree structure and the characteristics of nodes and leafs. This will allow him/her to take fine grain decisions regarding which resource carries out each task. Next, a set of these functions is shown. Note that the adopted prefix HMPI, that stands for hierarchical MPI, will be held in the rest of the document:

```
HMPI_GET_PARENT (hrank) returns hrank;
HMPI_GET_SONS (hrank) returns hrank [];
HMPI_GET_SIBLINGS (hrank) returns hrank [];
```

```
HMPI_GET_LEVEL(hrank) returns integer;
HMPI_GET_BANDWIDTH (hrank1, hrank2) returns float;
HMPI_GET_LATENCY (hrank1, hrank2) returns float;
```

Collective communications in the hierarchical model. We slightly modify the collective operations to take advantage of the existing hierarchy. Concretely we add a new parameter based on the associated hierarchy that indicates the group of processes involved in the operation. This modification is illustrated through the use of the reduce operation but is extended to the rest of collective operations:

`HMPI_REDUCE` (sendbuf, recvbuf, count, datatype, operation, root, HMPI_COMM_WORLD, hrank);

`Hrank` denotes here the root of the sub-tree which involved processes belong to. This gives the programmer the capability of easily communicating among process and synchronizing resources in different ways if they belong to different systems or are linked by different means.

Point to point communications in the hierarchical model. Elementary point to point operations – those involving a single sender and a single receiver – keep their semantics and only the way a process is identified changes as it has been exposed. These operations are called HMPI_SEND, HMPI_RECV, etc. In Figure 1, a former MPI_SEND (buf, count, datatype, 5, tag, MPI_COMM_WORLD) called by the process ranked 1, where MPI_COMM_WORLD is composed by the ranks showed on the row under the tree will be equivalent to HMPI_SEND (buf, count, data distribution, 0.1.0, tag, HMPI_COMM_WORLD) called by the process with hrank 0.0.1.

Nonetheless the most interesting feature is that this hierarchical representation enables using MPI communications at a higher level, going beyond the information exchange on a single process basis. For instance, suppose that in Figure 1, 0.0 sends some data to 0.1, in other words the processes 0.0.x call HMPI_SEND (buf, count, datadistribution, 0.1, tag, HMPI_COMM_WORLD). It involves a $N \times M$ communication and becomes transparent for programmers. All we need is to integrate data distribution support within the run-time environment, which is discussed in the next section.

Communications between any two levels are generalized straightforwardly once the environment knows how to manage the data distribution. Therefore, a tree node in level 1, as for instance 0.1, can send data to a tree node in level 2 as 0.1.0, as follows:

`HMPI_SEND` (buf, count, datadistribution, 0.1.0, tag, HMPI_COMM_WORLD) is called by all the processes 0.0.x.

`HMPI_RECV` (buf, count, datadescriptor, 0.0, tag, HMPI_COMM_WORLD, status) is called by 0.1.0.

It will likely result in some 0.0.x processes actually sending data to 0.1.0, while others will just wait for the end of the parallel point to point operation, depending on the information stored on datadistribution. Any parallel point to point operation may be, then, specified in the same terms.

4.2 Data distribution issues

The new communication type introduced above involves integrating the data model within the message passing framework and results in providing support and hiding complexity to programmers dealing with hierarchically partitioned data. Until now programmers were either obliged to care explicitly about scheduling the multiple low level communications or relying on a support library such as InterComm[8] or KeLP[9]. The goal is to allow the user to invoke a parallel send/receive operation through the hierarchy semantics with a high level view and disregarding distribution details. This facility entails making some modifications to the existing environment and how it manages information, so we treat these aspects next.

Support for managing distributed data. We propose to extend the MPI datatype with data distribution information, using an API to configure this distribution. The prior MPI datatype is to be improved with the distribution's precise description, including distribution type (blocks, regions, etc.), topology, partitioning information, overlapping areas, etc. Depending on the distribution type, these parameters are subject to modification. The lack of a generic API for setting up the environment with any given distribution leads us to propose different sets of functions specialized in each particular one.

Concretely, in the *DiscoGrid* context, descriptors need to store properties of multidimensional arrays that are hierarchically partitioned in blocks. Neighboring blocks share an overlapping area to exchange data between processes. In order to describe these aspects we gather two new features within MPI datatypes, that are configured through a set of operations: *Data Region and Interface Region.* Data region contains the distribution's dimensions, size and topological characteristics, and may be a subregion contained within a bigger one, in which case data structures of both will be linked. We provide, as well, an operation to set the mapping of data regions to resources so as to make the environment aware of the global distribution. Interface region enables the description of overlapping areas and, hence, describe the behavior of a parallel send/receive of the given distribution. These interface regions can be described as `IN` or `OUT` depending if they are aimed at receiving neighbor's data or at sending contained data to a neighbor. Therefore, the complete description of the distribution is made through a sequence of calls to the mentioned operations, resulting in a set of descriptors that are linked in a tree shape – as a result of the hierarchical partitioning.

As the framework is now aware of the data distribution and how it is assigned to computing resources, it is able to schedule the various low level messages needed to fulfill the hierarchical send/receive operations. If we deepen in this reasoning we can define a new operation at the top of this rich information context, enabling the user to simply call in parallel `UPDATE (buf, data_descriptor)` that makes, when called by all the processes associated to this data descriptor, all the necessary message passing sequence to correctly update the overlapping areas throughout the overall distribution. This mechanism strongly simplifies the application programming and makes the code much more legible.

4.3 Implementation issues

On the one hand target modifications intend to render the hierarchy of resources visible and easily manageable by application programmers. The necessary changes to prior MPI are related to adequately treating the hierarchy of resources and its associated `hranks`, which requires rich information about resource organization and properties. We should rely, thus, on the services provided by Information Systems specialized in Grid infrastructures that enable us to obtain information needed for managing the proposed environment. For instance, latency and bandwidth may be retrieved from a Network Weather Service [10].

On the other hand, the rich contents managed by this extended framework represent an extra amount of data structures to handle, but this just means transferring the workload to the framework and hence freeing the application programmer of such a task. The work to be done has mostly to do with scheduling issues in the distribution library, which is being widely researched as in InterComm or in our team's related project PaCO++, so we rely on a solid previous experience.

5 Conclusion

Massive computing resources are found more and more often organized as hierarchies of heterogeneously linked machines. The message passing model is being improved to take into account these new tendencies, so as to increase performance in such infrastructures. New hierarchical MPI versions enable programmers to take advantage of these powerful resources while providing a simple interface that hides important aspects of their inherent structures. Besides, they constitute a straightforward programming framework as they do not involve data distribution considerations.

However, some parallel applications, dealing with huge arrays and originally designed for a flat message passing model, appear less performing when tested on hierarchically distributed resources. They may be geared up in such execution environment by adapting data partitioning, and the communications this implies, with these hierarchical considerations. For this reason, the hierarchy of resources is to be made accessible and easy to manage within the applications, whereas handling of data distributions is to be supported by the same environment to enhance more efficient communications and code reusability. This is why we propose the MPI extension described in this report.

MPI's approach to hierarchies constitutes an important field of study with some major implementations, while the know-how regarding data distribution issues exists and it is being widely developed. The scientific applications outlined here can go one step further if relying on both techniques. In future work, main efforts concern the integration of these elements, for eventually providing a uniform framework for the described applications.

References

[1] Karonis, N., Toonon, B., Foster, I.: MPICH-G2: a Grid-enabled implementation of the Message Passing Interface. J. Parallel. Distrib. Comput. **63** (2003) 551–563

[2] Foster, I., Kesselman, C.: Globus: a metacomputing infrastructure toolkit. Int. J. Supercomput. Appl. **11** (1997) 115–128

[3] Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (2004) 97–104

[4] Matsuda, M., Kudoh, T., Kodama, Y., Takano, R., Ishikawa, Y.: Efficient mpi collective operations for clusters in long-and-fast networks. In: IEEE International Conference on Cluster Computing. (2006) 1–9

[5] Bertrand, F., Bramley, R., Sussman, A., Bernholdt, D.E., Kohl, J.A., Larson, J.W., Damevski, K.B.: Data redistribution and remote method invocation in parallel component architectures. In: IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers, Washington, DC, USA, IEEE Computer Society (2005) 40.2

[6] Prez, C., Priol, T., Ribes, A.: A parallel corba component model for numerical code coupling. The International Journal of High Performance Computing Applications (IJHPCA) **17** (2003) 417–429

[7] Forum, S.D.R.D.: Document for the data reorganization interface (dri-1.0). http://www.data-re.org (2002)

[8] Lee, J.Y., Sussman, A.: Efficient communication between parallel programs with intercomm (2004)

[9] S. Fink, S.K., Baden, S.: Efficient runtime support for irregular block-structured applications. J. Parallel. Distrib. Comput. **50** (1998) 61–82

[10] Wolski, R., Spring, N., Hayes, J.: The network weather service: A distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems **15** (1999) 757–768

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803