MPI Applications on Grids: A Topology-Aware Approach

Camille Coti, Thomas Herault, Franck Cappello

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

MPI Applications on Grids: a Topology Aware Approach

Camille Coti — Thomas Herault — Franck Cappello

N° 6633

Septembre 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6633--FR+ENG

MPI Applications on Grids: a Topology Aware Approach

Camille Coti , Thomas Herault , Franck Cappello

Thème NUM — Systèmes numériques
Équipe-Projet Grand Large

Rapport de recherche n° 6633 — Septembre 2008 — 18 pages

Abstract: Large Grids are built by aggregating smaller parallel machines through a public long-distance interconnection network (such as the Internet). Therefore, their structure is intrinsically hierarchical. Each level of the network hierarchy gives performances which differ from the other levels in terms of latency and bandwidth. MPI is the *de facto* standard for programming parallel machines, therefore an attractive solution for programming parallel applications on this kind of grid. However, because of the aforementioned differences of communication performances, the application continuously communicates back and forth between clusters, with a significant impact on performances. In this report, we present an extension of the information provided by the run-time environment of an MPI library, a set of efficient collective operations for grids and a methodology to organize communication patterns within applications with respect to the underlying physical topology, and implement it in a geophysics application.

Key-words: computational grids, message passing, collective operations, MPI

Centre de recherche INRIA Futurs
Parc Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 Orsay Cedex (France)
Téléphone : +33 1 72 92 59 00

Applications MPI pour la grille : une approche tenant compte de la topologie

Résumé : Les grilles de calcul à grande échelle sont construites en aggrégeant des machines parallèles plus petites et en les reliant à travers un réseau d'interconnexion longue distance public (comme l'Internet). Leur stucture est donc intrinsèquement hiérarchique. Chaque niveau de hiérarchie dans le réseau fournit des performances différentes des autres niveaux en termes de latence et de bande passante. MPI est le standard de fait pour programmer les machines parallèles, et donc une solution attractive pour programmer des applications parallèles pour ce type de grilles de calcul. Cependant, du fait des différences de performances de communications sus-citées, les applications communiquent entre les clusters avec un impact significatif sur les performances. Dans ce rapport, nous présentons une extension des informations fournies par l'environement d'exécution d'une bibliothèque MPI, un ensemble de communications collectives pour les grilles et une méthode pour organiser les schémas de communucations au sein de l'application en fonction de la topologie physique sous-jacente, et l'implémentons dans une application de gó-physique.

Mots-clés : grilles de calcul, passage de messages, opérations collectives, MPI

1 Introduction

Large Institutional Grids are built by aggregating smaller parallel machines (usually clusters) through a public interconnection network (such as the Internet). Because of their strong experience in parallel machines, many users wish to program such grids as traditional parallel machines although they do not know all the characteristics of such machines.

The *de-facto* standard for programming parallel machines is the Message Passing Interface (MPI). One of the advantages of MPI is that it provides a single well defined programming paradigm, based on explicit message passing and collective communications. It is interesting to consider a MPI for Grids, since complex applications may use non trivial communication schemes both inside and between clusters.

In order to port MPI on Grids, several issues have to be addressed. In [4], we already addressed the problem of designing an efficient runtime environment for MPI on Grids and enabling transparent inter-cluster communications. With the framework we designed in this preliminary work, an MPI application cannot take full advantage of the Grid power. Indeed, the communication topology does not differentiate communications between nodes inside a same cluster and remote nodes. Thus, in the general case, the application continuously communicates back and forth between clusters, with a significant impact on performances.

In this report we address the key problem of adapting the application's communication scheme to the topology of the grid on which the application is spawned. Our approach consists in specifying, during the application design and implementation, the topologies the application can be efficiently run on. In this specification, which will be described more thoughtfully in section 3, roles for specific nodes and network requirements between nodes can be defined.

A few approaches tried to tackle the topology adaptation problem (e.g. PACX-MPI and MPICH-G [8, 14]) by publishing a topology description to the application at runtime. However, practical experiments demonstrated that it is a difficult task to compute an efficient communication scheme from a topology discovered at runtime.

In the approach we develop inside the QosCosGrid european Project, the topology published at runtime to the application is compatible with the requirements of the application for an efficient data distribution, and communications. Moreover, the topology published contains additional information on nodes and networks thus allowing to specialize roles for specific nodes.

Furthermore, topology information can be used to organize point-to-point communications within collective operations. Collective operations have been studied and optimized during the last decades under a common assumption: the communication cost function between two nodes is the same throughout the system. This hypothesis is no longer valid in a grid environment, since communication costs can vary by three orders of magnitude between two consecutive levels of hierarchy. Hence, performances can greatly benefit from collective operations that adapt their point-to-point communications to the topology.

The main contributions of this article are: a grid-enabled MPI middleware, featuring topology discovery functionality; a set of adapted collective operations that fit with the topology of the grid using topology information, namely Broadcast, Reduce, Gather, Allgather, Barrier and Alltoall; a grid-enabled application that takes advantage of the aforementioned features.

2 Related Work

Open MPI[5] is the MPI implementation used as the basis of the grid-enabled middleware [4] of the QosCosGrid project [16]. It fits particularly well to grid computing, because it allows heterogeneous 32/64 bits representation, and can handle more complex shifting between different architectures.

The PACX-MPI[8] project was created to interconnect two vendor parallel machines. It can be used to aggregate several clusters and create a meta-computer.

Grid-MPI [17] is a grid-enabled extension of YAMPII using the IMPI [9] standard. It implements the MPI-1.2 standard and most of the MPI-2 standard. In [17] some optimized collective operations are presented for two clusters. The *AllReduce* algorithm is based on the works presented in [20]. The broadcast algorithm is based on [1]. Collective operations are optimized to make an intensive use of inter-cluster bandwidth, with the assumption that inter-cluster communications have access to a higher bandwidth than intra-cluster. However, this is not always a valid assumption. Nowadays, high-speed and low latency networks' bandwidth is higher than inter-cluster Internet links. An environment where the inter-cluster bandwidth is lower than the intra-cluster bandwidth is out-of-scope of [17]. Furthermore, GridMPI does not allow the use of heterogeneous environments because of different precision in floating point and 32/64 bits support.

The Globus Toolkit [7] is a set of software that aims to provide tools for an efficient use of grids. MPICH [10] has been extended to take advantage of these features [14] and make an intensive use of the available resources for MPI applications. It uses GT for resource discovery, staging, spawning, launching and to monitor the application, input and output redirection and topology discovery. MPICH-G2 introduced the concept of colors to describe the available topology. It is limited to at most four levels: WAN, LAN, system area and, if available, vendor MPI. Those four levels are usually enough to cover most use-cases. However, one can expect finer-grain topology information and more flexibility for large-scale grid systems. Besides, colors in MPICH-G2 are integers only. This is convenient to create new communicators using MPI_Comm_split(), but the user has to "guess" what color corresponds to the requested communicator.

All of these approaches display the *physical* topology for the application. On the other hand, our approach focuses on communication pattern: the application has access to the *logical* topology, which maps the communications of the application. Another element of the grid architecture, the meta-scheduler, which is not described in this work, is in charge with matching this requested logical topology with a compatible physical topology.

Collective operations have been studied widely and extensively the last decades. One can cite Fibonacci (k-ary) trees and binomial trees for broadcast operations. Binary trees can be optimized as split-binary trees. The message is split equally by the root [23], and each half is sent to each side of the tree. At the end of the first phase, each half of the nodes has received half of the message. Each node exchanges its half with a node among the other half.

However, as pointed out in [20] those strategies are optimal in homogeneous environments, and most often with a power-of-two number of processes. Their performance are drastically harmed by non-power-of-two numbers of processes and heterogeneous message transmission times.

Topology-discovery features in Globus have been used to implement a topology-aware hierarchical broadcast algorithm in MPICH-G2 [13]. The basic idea of this study is to minimize inter-cluster communications. Instead of sending $\log(p)$ messages between two clusters (p being the overall number of nodes in the system), as a traditional binary tree would, the algorithm presented in this article sends only one message.

A hierarchical broadcast algorithm is also presented in [3], with a very detailed communication cost model and an deep analysis of the problem. The model used in this study considers not only heterogeneity in terms of communication delays but also in terms of message processing speed. It considers a multi-level strategy to minimize the complexity of an instance of the Weighted Broadcast Problem with an extension of a Greedy Broadcasting Algorithm.

An optimized reduction algorithm for heterogeneous environments is presented in [15]. However, the model used in that study considers only the time each node takes to process a message, not point-to-point communication times.

Wormhole is a routing protocol targeted to massively parallel computers. Messages are divided into small pieces called *flits*, which are pipelined through the network. [18] studies how collective operations can benefit or be harmed by message splicing.

3 Architecture

To implement our topology aware approach, we have designed QCG-OMPI, a modified version of Open MPI for the QosCosGrid project. The general architecture of QCG-OMPI is detailed in [4]. A set of persistent services extend the runtime environment of the parallel application in order to support applications spanning above several clusters.

These services are called Inter-Cluster Communication Services (ICCS). They make possible peer-to-peer connections between nodes separated by a firewall and/or a NAT. The ICCS architecture is depicted in figure 1.

Several inter-cluster connection techniques are implemented in QCG-OMPI. Basic techniques include direct connection (when possible according to the firewall policy of clusters), relaying (like PACX-MPI, though with only one relay daemon instead of one for incoming communications and one for outgoing ones), limitation to an open port range (like MPICH-G2 and PACX-MPI) and reverse connection. More elaborate methods include traversing TCP, which is presented in [21].

These features are provided by a hierarchical distributed grid infrastructure consisting on a set of grid services. A *connection helper* is running on every node to help MPI processes establish connections with other processes. A *frontal* service is running on the front-end machine of every cluster. It is used for message relaying within the infrastructure, and caches information. The only centralized service is the *broker*. It maintains an up-to-date global contact list about all processes of the applications and knows which is the most efficient technique to interconnect two processes.

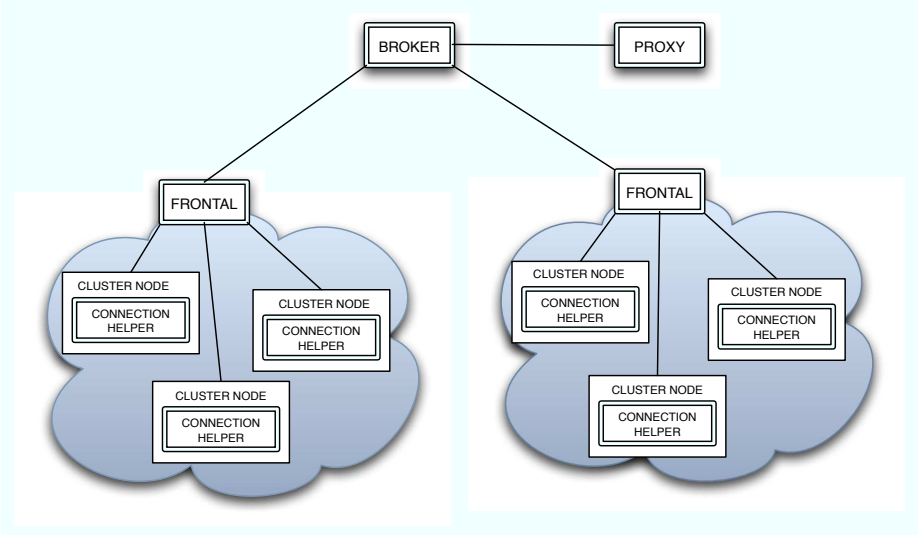


Figure 1: Inter-cluster communication services

3.1 Multi-level topology discovery

Institutional grids are made of several levels of hierarchy: nodes can feature multi-cores or multi-processors architectures, they are put together as clusters (sharing a same network), sometimes several clusters are geographically close to each other (*e.g.*, in the same machine room or in the same building), yet cannot be considered as a single cluster, and clusters located in different organizations are interconnected through the Internet. Hence, applications ought to be adapted to the underlying topology, in order to consider it to organize their communications. However, the MPI standard does not specify such "geographical" information access.

We propose to describe the topology by naming the groups of nodes the processes will be mapped on. This information is passed as a parameter of the scheduler to specify hardware requirements on the requested nodes.

We give the developer the possibility to describe a topology for his application. Beside the application itself, he can write a *JobProfile* describing requirements in terms of communication possibilities and/or computational power for the application processes. A meta-scheduler is in charge of mapping the requested topology on available nodes whose characteristics match as tightly as possible the ones requested in the JobProfile.

This approach allows the application developer to organize his application (and more specifically his communications) with in his mind the physical topology the application will be executed on. An example of a minimal jobProfile is presented figure 2. It describes the process groups involved in the computation (only one group here), in particular by specifying the number of processes in each group (here $NPROCS, *i.e.*, all the available processes in the job). Some parameters are left blank, and filled by the meta-scheduler with the characteristics of the obtained mapping.

```
<grmsJob appId="helloworld-job">
  <task taskId="helloworld-task1">
    <resourcesDescription>
      <topology>
        <group groupId="g1"/>
        <processes>
          <groupIdReference groupId="g1"/>
          <processesCount>
            <value>$NPROCS</value>
          </processesCount>
        </processes>
      </topology>
    </resourcesDescription>
    <execution type="open_mpi">
      <executable>
        <execFile>
          <file>
            <url>gsiftp://host.domain.com/~/hello</url>
          </file>
        </execFile>
      </executable>
      <processesCount>
        <value>$NPROCS</value>
      </processesCount>
    </execution>
  </task>
</grmsJob>
```

Figure 2: Simple JobProfile example

The *groupId* defined in the jobProfile is the same as the one obtained by the application in the topology description. This way it is easy to determine in the application which group a given process belongs to, and which processes belong to a given group. Furthermore, a bijective function is provided by the MPI library to get the color integer corresponding to the name of the group, to provide ability to create communicators.

The following two subsections explain how to take advantage of those colors. Subsection 3.2 describe a set of collectives operations designed to fit on the physical topology and knowledge about proximity between processes, and subsection 3.3 describes a specifically adapted application.

The difference beteewn the physical and the logical topology it that the latter is defined by the developper, and the former is obtained by the scheduler and matches with the logical topology. For example, if the developper requested three groups for tightly coupled processes, the scheduler can map them on two clusters only: the physical topology meets the requirements of the logicial topology, but is not exactly the same as what was specified. Global collective operations must consider the whole set of processes as two subsets, whereas the application considers it as three subsets.

3.2 Adapted collective operations

Collective operations are critical in MPI applications. A study conducted at the Stuttgart High-Performance Computing Center [19, 20] showed that on their Cray T3E, they represent 45% of the overall time spent in MPI routines.

To the best of our knowledge, no equivalent study was ever done on a production grid during such a long period. However, one can expect non-topology-aware collective communications to be even more time-consuming (with respect to all the other operations) on an heterogeneous platform.

```
// communicators are in a table of MPI_Comm
QCG_Bcast(buffer, ROOT, **comms){
 for( i = 0 ; i < depth[my_rank] ; i++) {
        MPI_Bcast(&buffer, 1, MPI_INT, ROOT,
                  comm[i] );
 }
}
QCG_Barrier(**comms){
 for( i = depth[my_rank]-1 ; i >= 0 ; i-) {
        MPI_Barrier( comm[i] );
 }
 QCG_Bcast(&i, 1, MPI_INT, ROOT, &comm);
}
```

Figure 3: Hierarchical broadcast and barrier

Usage of topology information In order to use the topology information for collective operations, we define the following groups of processes and the corresponding communicators: a) Each topological group (*e.g.*, a cluster) is assigned a communicator; b) Among each of those groups, a master is defined (*e.g.*, the lowest global rank); c) All the processes within a given group that are masters of their respective subgroups are part of a master group

MPI_Bcast Sending a message between two clusters takes significantly more time than sending a message within a cluster. The latency for small synchronization messages, can be superior by several orders of magnitude, and the inter-cluster bandwidth is shared between all the nodes communicating between clusters. Thus, a priority of any collective algorithm in grid environments is to minimize inter-cluster communications.

In this study, we use a hierarchical broadcast. Most often, broadcasts are initiated by the process of rank 0 of the concerned communicator. The root of the broadcast, being part of the top-level master communicator, broadcasts the message along this top-level communicator. Each process then broadcasts the message along its "sub-masters" communicator, until the lowest-level nodes are reached.

The system of colors does not allow the processes to know about communication costs within a level, so we assume here that each sub-group is homogeneous.

MPI_Reduce Reduction algorithms are not just "reverse" broadcasting algorithms. Indeed, the operation may not be commutative, although all the predefined operators are commutative. However, it is required to be associative.

We assume that the processes mapped in a given cluster have consecutive ranks. Therefore, we can use the hierarchical approach for a reduction operation too. Each lowest level cluster perform a reduction towards their master, and for each level until the top level is reached the masters perform a reduction toward their level master.

MPI_Gather A *Gather* algorithm sends the contents of a buffer from each process to the root process of the gather operation. It can also be done in a hierarchical way: a root is defined in each cluster and sub-cluster, and an

optimized gather algorithm is used within the lowest level of hierarchy, then for each upper level until the root is reached.

The executions among sub-masters gather buffers which are actually aggregations of buffers. This aggregation minimizes the number of inter-cluster communications, for the cost of only one trip time while making a better use of the inter-cluster bandwidth.

MPI_Barrier *blocks the caller until all group members have called it. The call returns at any process only after all group members have entered the call.* [22]. This fact can be guaranteed by successive MPI_Barrier from the lowest-level communicator up to the highest-level masters' communicator. At this point, all the lowest-level masters are sure that all the callers of MPI_Barrier entered the barrier, then spread the information using a hierarchical broadcast, refered as QCG_Bcast in figure 3.

MPI_Allgather can be achieved using the same algorithm as MPI_Barrier.

MPI_Alltoall [12] presents an algorithm for all-to-all communications on clusters of clusters. Nodes within each cluster perform an all-to-all communication, then several nodes in each cluster send an aggregation of several buffers to nodes belonging to the other clusters. These meta-buffers are broadcast within the other clusters.

The goal of this cutting-through is to share the quantity of data that must be sent between two given nodes. However, if the meta-buffers do not contain enough data to fill the inter-cluster bandwidth, it will not fully take advantage of those links.

Applications do not usually send extremely large data through MPI communications ; hence, the meta-buffer containing the result of each local all-to-all will be sent in only one communication.

To summarize, an all-to-all communication is first performed within each lowest-level set of processes. Then at each level, the masters perform an all-to-all communication with aggregation of the buffers sent in the lower level. When the top-level is reached, the result is broadcast (using a topology-aware broadcast algorithm).

3.3 Grid-enabled application

We implemented and used these collective operations in Ray2mesh [11], a geophysics application that traces seismic rays along a given mesh containing a geographic area description. It uses the Snell-Descartes law in spherical geometry to propagate a wave front from a source (earthquake epicenter) to a receiver (seismograph). It features two parallel implementations, including a master-worker-based approach.

The execution of Ray2mesh can be split up into three phases. The first one consists of successive collective operations to distribute information to the computation nodes. It is made of three broadcast operations, from the master toward workers. Broadcasts are tree-based collective operations: calls return as soon as the process's participation to the operation is completed, not when all the operation is completed. For example, the first stage of a tree-based operation

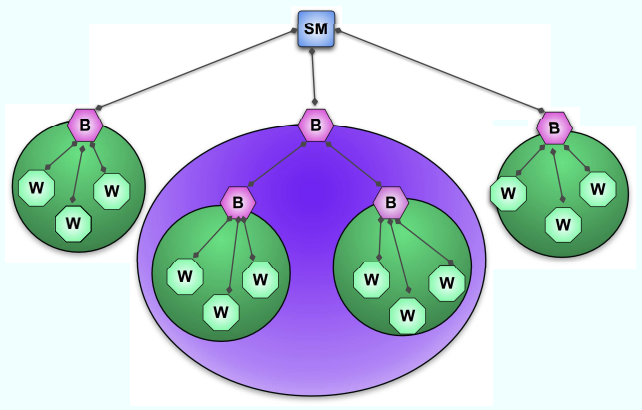


Figure 4: Organization of a multi-level master-worker application. SM is the Super Master, B are the Bosses, W are the Workers

can start as soon as the second stage of the previous one starts. Operations performed here between broadcasts are simple operations, like memory allocations: the broadcasts are pipelined. The second phase is the master-worker computation itself. The third phase is made of collective operations: a broadcast and an all-to-all operation.

The master-worker approach is used for a very wide range of parallel applications. Its major drawback is the concentration of all the data distribution work in the master, creating a bottleneck. Nevertheless, the unique queue (central master) design is a basic approach of a master-worker data distribution.

Moreover, the master can be located in a remote cluster. After computing a chunk of data, slaves wait for a new chunk of data: the longer it takes to send the result to the master and to receive a new chunk, the longer the worker will be idle. Data prefetch to overlap communication and computation [2] can be a solution to eliminate those idle phases, however this approach requires massive modification of an application and complexifies the utilisation of generic libraries.

Our approach is focused on communication patterns. We use the topological information to build a multi-level implementation of the three phases involved in Ray2mesh to make the communication pattern fit with the topology of the Grid.

Processes which are close to each other can communicate often, whereas distant processes ought to barely (or not at all) communicate with each other. We organize communications with this basic principle in mind. Processes located in the same cluster (we consider as "cluster" a set of processes that share the same lowest-level color) share the same data queue. A process within each cluster is in charge to distribute the data among other processes: we call it the local *boss*. Then, workers receive data from their local boss, involving local-only communications.

Bosses get their data queue from an upper-level boss (that can be the top-level master, or *super-master*). Therefore, most communications are done within clusters. The most expensive communications in terms of latency, between two top-level clusters, are the least frequent ones. They are used to transfer larger

sets of data, destinated to bosses: they make a better use of a (supposed) wider bandwidth and reduce the effects of a higher latency.

The organization of a multi-queue master-worker application is represented in figure 4, with 2 clusters and 2 sub-clusters in one cluster.

This approach provides the same attractive properties as a traditional master-worker application, with any number of levels of hierarchy. Hence, it performs the same automatic load-balancing, not only among the workers, but also among the queues: if a cluster computes its data faster than the other ones, it will get a new set of data immediately after the results are recovered. This property is necessary to suit to different sizes of clusters and different computation speeds. Moreover, each master has to handle fewer work requests than a unique master would have to.

Beside of this master-worker computation, Ray2mesh uses several collective operations. Some information related to the computation is broadcast before the actual master-worker phase: our grid-enabled broadcast algorithm is used three times before the computation phase, then once after it. Some information about the results computed by each slave are eventually sent to each other: we use here our hierarchical MPI_Alltoall.

4 Experimental Evaluation

In this section we present the performance measurements of our implementation. We conducted the experiments on two traditional platforms of high performance computing: clusters of workstations with GigaEthernet network and computational grids. These experiments were done on the experimental Grid'5000 [6] platform or some of its components.

First, we measure the efficiency of topology-aware collective operations, using micro-benchmarks to isolate their performance. Then we measure the effects of hierarchy on a master-worker data distribution pattern and the effects on the Ray2mesh application.

4.1 Experimental Platform

Grid'5000 is a dedicated reconfigurable and controlable experimental platform featuring 13 clusters, each with 58 to 342 PCs, connected by Renater (the French Educational and Research wide area Network)."

It gathers roughly 5000 CPU cores featuring four architectures (Itanium, Xeon, G5 and Opteron) distributed into 13 clusters over 9 cities in France.

For the two families of measurement we conducted (cluster and grid), we used only homogeneous clusters with AMD Opteron 248 (2 GHz/1MB L2 cache) bi-processors. This includes 3 of the 13 clusters of Grid'5000: the 93-nodes cluster at Bordeaux, the 312-nodes cluster at Orsay, a 99-nodes cluster at Rennes. Nodes are interconnected by a Gigabit Ethernet switch.

We also used QCG, a cluster of 4 multi-core-based nodes with dual-core Intel Pentium D (2.8 GHz/2x1MB L2 cache) processors interconnected by a 100MB Ethernet network.

All the nodes were booted under linux 2.6.18.3 on Grid'5000 and 2.6.22 on the QCG cluster. The tests and benchmarks are compiled with GCC-4.0.3 (with flag -O3). All tests are run in dedicated mode.

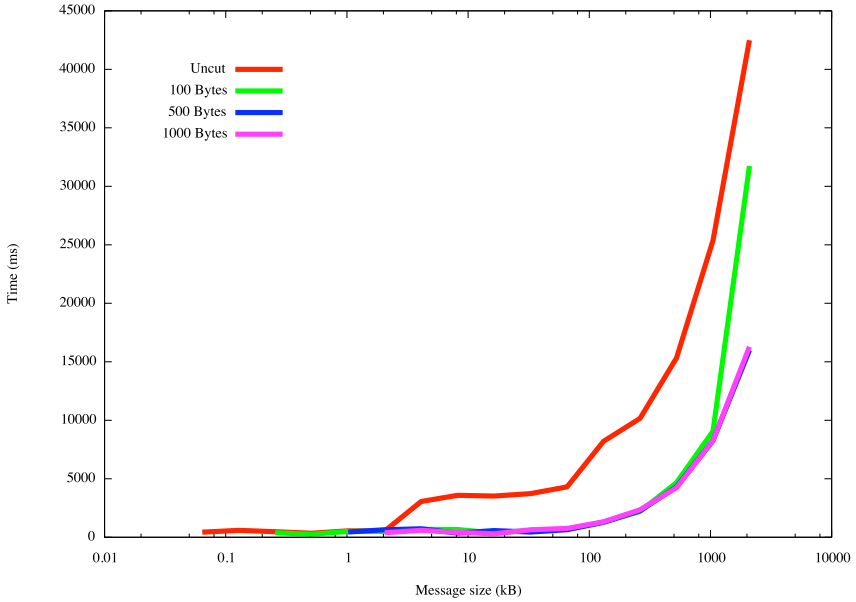


Figure 5: Hierarchical MPI_Gather

Inter-cluster throughput on Grid'5000 is 136.08 Mb/s and latency is 7.8 ms, whereas intra-cluster throughput is 894.39 Mb/s and latency is 0.1 ms. On the QCG cluster, shared-memory communication have a throughput of 3979.46 Mb/s and a latency of 0.02 ms, whereas TCP comminucations have a throughput of 89.61 Mb/s and a latency of 0.1 ms.

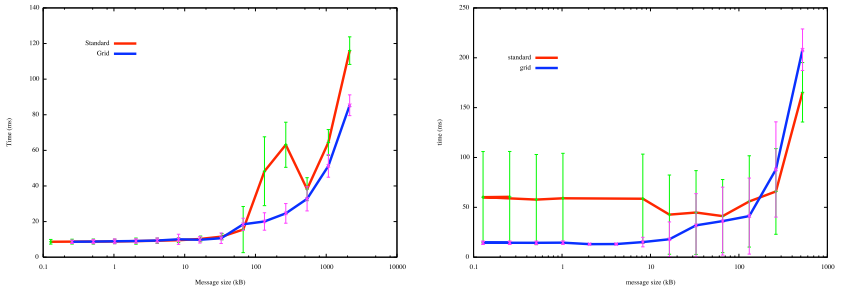
4.2 Collective operations

We implemented the collective operations described in section 3.2 using MPI functions. We used them on 32 nodes across two clusters in Orsay and Rennes (figures 6a-b). Although not exactly a "grid" in the sense of a cluster of clusters, a configuration with two clusters is an extreme situation to evaluate our collective communications: a small constant number of inter-cluster messages are sent by topology-aware communications, whereas $O(log(p))$ (p: total number of nodes) inter-cluster messages are sent by standard collective operations. With any number of clusters, topology-aware collective operations send $O(log(C))$ inter-cluster messages, where C is the number of clusters.

We also used the QCG cluster with 8 processes mapped on each machine. Although this oversubscribes the nodes (8 processes for 2 available slots), our benchmarks are not CPU-bounded, and this configuration enhances the stress on the network interface. Measurements with a profiling tool validated the very low CPU usage during our benchmark runs.

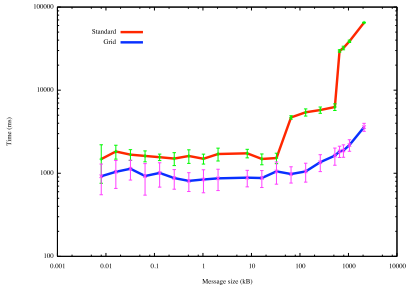
We used the same measurement method as described in [14], using the barrier described in section 3.2.

Since we implemented our hierarchical collective operations in MPI, some pre-treatment of the buffers may be useful. We compared the performances of our hierarchical MPI_Gather for different message size on the QCG cluster

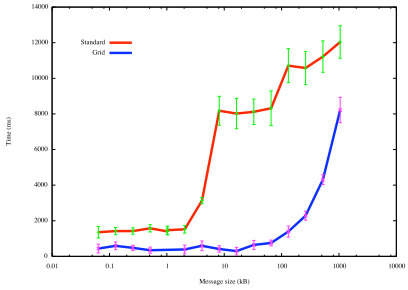


(a) Broadcast

(b) Reduce



(c) Reduce



(d) Gather

Figure 6: Comparison between standard and grid-enabled collective operations on a grid.

figure 5. Pre-splicing the buffer appeared to be interesting for messages larger than 1kB. Then it is possible to pipeline the successive stages of the hierarchical operation. It appeared to be useful especially when shared-memory communications were involved, which can be explained by the better performances of shared-memory communications on small messages.

Figures 6(a) and 6(b) picture comparisons between standard and hierarchical MPI_Bcast and MPI_Reduce on Grid'5000. Splicing appeared to be useful for MPI_Bcast: we spliced the messages every 10 kB for messages of size 64kB and beyond, whereas it was useless for MPI_Reduce, since big messages are already cut out by the algorithm implemented in Open MPI.

One can see that, as expected, hierarchical MPI_Bcast (figure 6(a)) always performs better than the standard one. Moreover, splicing and pipelining permits to avoid the performance step around the eager/rendezvous mode transition.

When messages are large regarding the communicator size, MPI_Reduce (figure 6(b)) in Open MPI is implemented using a pipeline mechanism. This way, the communication cost is dominated by the high throughput of the pipeline rather than the latency of a multi-steps tree-like structure. Hierarchy shortens the pipeline: then its latency (*i.e.*, time to load the pipeline) is smaller and it performs better on short messages. But for large messages (beyond 100 kB), the higher throughput of a longer pipeline outperforms the latency-reduction strategy. In this case, hierarchization of the communications is not the appropriate approach. The high standard deviation of the standard MPI_Reduce for small

messages can be explained by the high variability of concurrent traffic on the inter-cluster links, and the fact that the standard algorithm for small messages is more sensitive to this.

Figures 6(c) and 6(d) picture comparisons between standard and hierarchical MPI_Reduce and MPI_Gather on the QCG cluster. On a cluster of multi-cores, collective operations over shared-memory outperform TCP communications significantly enough to have a negligible cost. Therefore, on a configuration including a smaller number of physical nodes, inducing more shared-memory communications, our hierarchical MPI_Reduce performs better (figure 6(c)).

For the same reasons, MPI_Gather performs better if communications are organized in order to fit with the physical topology. Moreover, performances seem to be drastically harmed by the transition to the rendezvous communication protocol, which is by default set to 4 kB in Open MPI. We cut messages out every 1 kB for the hierarchical mode, as explained in figure 5.

4.3 Adapted application

The execution phases of Ray2mesh are presented in section 3.3. It is made of 3 phases: 2 communication phases and a master-worker computation phase in between them. When the number of processes increases, one can expect the second phase to be faster but the first and third phases to take more time, since more nodes are involved in the collective communications.

Figure 7 presents the scalability of Ray2mesh under three configurations: standard (vanilla), using grid-enabled collective operations, and using a hierarchical master-worker pattern and grid-enabled collective operations. Those three configurations represent the three levels of adaptation of applications to the Grid. The standard deviation is lower than 1% for each point.

First of all, Ray2mesh scales remarkably well, even when some processes are located on a remote cluster. But performances are slightly better on a single cluster, for obvious reasons: collective operations are faster on a cluster, and workers stay idle longer on a grid while they are waiting for a new set of data.

When a large number of nodes are involved in the computation, collective operations represent an important part of the overall execution time. We can see the improvement obtained from grid-enabled collectives on the "grid-optimized collectives" line in figure 7. The performance gain for 180 processes is 9.5%.

Small-scale measurements show that the grid-enabled version of Ray2mesh does not perform as well as the standard version. The reason for that is that several processes are used to distribute the data (the bosses) instead of only one. For example, with 16 processes distributed on two clusters, 15 processes will actually work for the computation in a single-queue master-worker application, whereas only 12 of them will contribute to the computation on a multi-level (two-level) master-worker application.

However, we ran processes on each of the available processors, regardless of their role in the system. Bosses are mainly used for communications, whereas workers do not communicate a lot (during the master-worker phase, they communicate with their boss only). Therefore, a worker process can be run on the same slot as a boss without competing for the same resources. For a given number of workers, as represented by the "workers and master only" line in figure 7, the three implementations show the same performance for a small number of

Figure 7: Scalability of Ray2mesh on a grid

processes, and the grid-enabled implementations are more scalable. The performance gain for 180 processes is 35%.

5 Conclusion

In this report, we presented an extension of the runtime environment of an MPI implementation targeting institutional grids to provide topology information to the application. Those features have been implemented in our specific adaptation of an MPI library for grids.

We proposed a methodology to use MPI programming techniques on grids. First we described a set of efficient collective operations that organize their communication with respect to the topology in order to minimize the number of high-latency communications. Experiments showed the benefits of this approach and its limitations.

Then we presented a programming method to make use of the topology information in the application and define adapted communication patterns. We used this method to implement a grid-enabled version of the Ray2mesh geophysics applications featuring a multi-level master-worker pattern and our hierarchical collective operations. Experiments showed that using optimized collectives fitted to the physical topology of the Grid induce a performance improvement. They also showed that adapting the application itself can improve the performances even further.

We are now working on the integration of our grid-enabled MPI middleware with the Distributed Resource Management Application API (DRMAA) OpenDSP [24]. OpenDSP features job submission, job termination and job monitoring services, and provides a dedicated authentication services to increase the level of security and confidentiality across the grid.

Acknowledgments

The authors thank Emmanuel Jeannot for the interesting discussions about collective operations, and Stephane Genaud and Marc Grunberg for their help on Ray2mesh. A special thank must be given to George Bosilca for his explanations about the implementation of collective operations in OpenMPI.

Part of the authors are founded through the QosCosGrid European Project (grant number: FP6-2005-IST-5 033883).

References

[1] Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Building a high-performance collective communication library. In *Proceedings of Supercomputing'94*, pages 107–116, Washington DC, November 1994. IEEE.

[2] Salah-Salim Boutammine, Daniel Millot, and Christian Parrot. An adaptive scheduling method for grid computing. In Wolfgang E. Nagel, Wolfgang V. Walter, and Wolfgang Lehner, editors, *12th International Euro-Par Conference (Euro-Par'06)*, volume 4128 of *Lecture Notes in Computer Science (LNCS)*, pages 188–197, Dresden, Germany, August-September 2006. Springer-Verlag (Berlin/New York).

[3] Franck Cappello, Pierre Fraigniaud, Bernard Mans, and Arnold L. Rosenberg. HiHCoHP: Toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (2nd IPDPS'01)*, page 42, San Francisco, CA, USA, April 2001. IEEE Computer Society (Los Alamitos, CA).

[4] Camille Coti, Thomas Herault, Sylvain Peyronnet, Ala Rezmerita, and Franck Cappello. Grid services for MPI. In ACM/IEEE, editor, *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08)*, Lyon, France, May 2008.

[5] E. Gabriel *et al*. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[6] F. Cappello *et al.* Grid'5000: a large scale, reconfigurable, controlable and monitorable grid platform. In *proceedings of IEEE/ACM Grid'2005 workshop*, Seattle, USA, 2005.

[7] Ian T. Foster. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci. Technol*, 21(4):513–520, 2006.

[8] Edgar Gabriel, Michael M. Resch, Thomas Beisel, and Rainer Keller. Distributed computing in a heterogeneous computing environment. In Vassil N. Alexandrov and Jack Dongarra, editors, *PVM/MPI*, volume 1497 of *Lecture Notes in Computer Science*, pages 180–187. Springer, 1998.

[9] William L. George, John G. Hagedorn, and Judith E. Devaney. IMPI: Making MPI interoperable, April 25 2000.

[10] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[11] Marc Grunberg, Stephane Genaud, and Catherine Mongenet. Parallel seismic ray tracing in a global earth model. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA'02)*, volume 3, pages 1151–1157, Las Vegas, Nevada, USA, June 2002. CSREA Press.

[12] E. Jeannot and Luiz-Angelo Steffenel. Fast and efficient total exchange on two clusters. In *the 13th International Euro-Par Conference*, volume 4641 of *LNCS*, pages 848–857, Rennes, France, August 2007. Springer Verlag.

[13] N. T. Karonis, B. de Supinski, I. Foster, B. Gropp, E. Lusk, and T. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *IPPS: 14th International Parallel Processing Symposium*. IEEE Computer Society Press, 2000.

[14] Nicholas T. Karonis, Brian R. Toonen, and Ian T. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *CoRR*, cs.DC/0206040, 2002.

[15] Liu and Wang. Reduction optimization in heterogeneous cluster environments. In *IPPS: 14th International Parallel Processing Symposium*. IEEE Computer Society Press, 2000.

[16] M. Charlot *et al.* The QosCosGrid project: Quasi-opportunistic supercomputing for complex systems simulations. Description of a general framework from different types of applications. In *Ibergrid 2007 conference, Centro de Supercomputacion de Galicia (GESGA)*, 2007.

[17] Motohiko Matsuda, Tomohiro Kudoh, Yuetsu Kodama, Ryousei Takano, and Yutaka Ishikawa. Efficient MPI collective operations for clusters in long-and-fast networks. In *CLUSTER*. IEEE, 2006.

[18] McKinley, Tsai, and Robinson. Collective communication in wormhole-routed massively parallel computers. *COMPUTER: IEEE Computer*, 28, 1995.

[19] Rolf Rabenseifner. Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512, January 25 1999.

[20] Rolf Rabenseifner. Optimization of collective reduction operations. In Marian Bubak, G. Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *International Conference on Computational Science*, volume 3036 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2004.

[21] Ala Rezmerita, Tangui Morlier, Vincent Néri, and Franck Cappello. Private virtual cluster: Infrastructure and protocol for instant grids. In Wolfgang E. Nagel, Wolfgang V. Walter, and Wolfgang Lehner, editors, *Euro-Par*, volume 4128 of *Lecture Notes in Computer Science*, pages 393–404. Springer, 2006.

[22] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996.

[23] Rajeev Thakur and William Gropp. Improving the performance of collective operations in MPICH. In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, *PVM/MPI*, volume 2840 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2003.

[24] Peter Tröger, Hrabri Rajic, Andreas Haas, and Piotr Domagalski. Standardization of an API for distributed resource management systems. In *CCGRID*, pages 619–626. IEEE Computer Society, 2007.

Centre de recherche INRIA Futurs
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399