MPI Applications on Grids: a Topology Aware Approach

Camille Coti† Thomas Herault‡† Franck Cappello†
coti@lri.fr *herault@lri.fr* *fci@lri.fr*
†*INRIA, F-91893 Orsay France*
‡*Univ Paris Sud; LRI; F-91405 Orsay France*

Abstract

Porting on grids complex MPI applications involving collective communications requires significant program modification, usually dedicated to a single grid structure. The difficulty comes from the mismatch between programs organizations and grid structures: 1) large grids are hierarchical structures aggregating parallel machines through an interconnection network, decided at runtime and 2) the MPI standard does not currently provide any specific information for topology-aware applications, so almost all MPI applications have been developed following a non-hierarchical and non-flexible vision. In this paper, we propose a generic programming method and a modification of the MPI runtime environment to make MPI applications topology aware. In contrary to previous approaches, topology requirements for the application are given to the grid scheduling system, which exposes the compatible allocated topology to the application.

1 Introduction

Porting MPI applications on grids and getting acceptable performance is challenging. However two clear user motivations push researchers to propose solutions: 1) The *defacto* 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 an MPI for grids, since complex applications may use non trivial communication schemes both inside and between clusters; 2) Because of their experience in parallel machines, many users wish to port their existing MPI applications, but redeveloping large portions of their codes to fit new paradigms requires strong efforts.

Not all parallel applications will perform well on a grid, and in general optimizations are required to reach acceptable performance. However, computation intensive applications following the master-worker or monte-carlo approaches are good candidates and some of them have been ported and executed successfully on grids [1, 19, 4].

In this paper, we investigate the issue of porting more complex MPI applications on grids. More specifically, we consider applications involving some collectives communications. In order to port complex MPI applications on grids, several issues have to be addressed. In [8], we already addressed the problem of designing an efficient MPI on grids and enabling transparent inter-cluster communications. However, with this framework, MPI applications cannot take full advantage of the grid performance. Indeed, the communication pattern does not differentiate communications between nodes inside a cluster and remote nodes. As a consequence, the application may continuously communicate between clusters, with a significant impact on performances.

The difficulty of porting complex MPI applications on grids comes from 1) the difference between MPI programs organization and grid structures and 2) the static organization of existing MPI programs that does not fit with the diversity of grid structures. Cluster of clusters grids are intrinsically hierarchical structures where several parallel machines are connected through a long-distance interconnection network. In contrary, MPI standard does not currently provide any specific information on the topology, so almost all MPI applications have been developed following a non-hierarchical vision. In addition all grids differ in their topology and there is no mechanism in MPI to self-adapt the topology of the application to the one of the execution environment.

Previous works addressed the lack of topology awareness in MPI by exposing the topology of the available resources to the application. However, this approach requires a strong effort from the application to adapt itself to potentially any kind of resources that can be available at the time of submission, and is a key lock to building topology-aware MPI applications for grids. Such applications need to have a generic computation pattern that can adapt itself to any communication pattern, and such applications are very difficult (and sometimes impossible) to program. Our approach to address this problem is to combine: a) a modification of the MPI program organization to make it hierarchical and flexible b) a description by the programmer of its hierarchical communication pattern through a virtual topology and c) a mapping of the virtual topology to the physical one as provided by the grid reservation and scheduling service.

Typically in our approach, the application developer adapts the application code in a hierarchical approach and describes its ”virtual” computation and communication patterns in a companion file. The application developer specifies in the companion file properties for specific processes and network requirements between nodes. To execute the application, the user submits it to the grid by providing as usual the binary, its parameters and data files, the number of desired nodes, and the companion file. We assume that the grid reservation and scheduling system assigns physical resources to the application according to a best effort matching with the user requirements and the application’s companion file. This assumption corresponds to the architecture proposed in the QosCosGrid project, and scheduling techniques to allocate resources corresponding to the developer and user requirements are described in [7, 16, 17]. The modified MPI system adapts the collective operations to optimize communications on the physical topology, and exposes the virtual topology required by the developer to the application, thus optimizing communication patterns to the hierarchical topology. Since communication costs can vary by orders of magnitude between two consecutive levels of topology hierarchy, performances can greatly benefit from collective operations that adapt their point-to-point communications pattern to the physical topology.

We present 4 main contributions to expose our approach in details and demonstrate its effectiveness: 1) the method to make MPI applications adapt to grids’ hierarchy; 2) the presentation and performance evaluation of a grid-enabled MPI middleware, featuring topology awareness; 3) the evaluation of adapted collective operations that fit with the topology of the grid using topology information, namely Broadcast, Reduce, Gather, Allgather and Barrier; 4) the description and evaluation of a grid-enabled application that takes advantage of our approach.

2 Related Work

A few approaches tried to tackle the topology adaptation problem (e.g. PACX-MPI and MPICH-G [10, 14]) by publishing a topology description to the application at runtime. The Globus Toolkit (GT) [9] is a set of software that aims to provide tools for an efficient use of grids. MPICH [11] has been extended to take advantage of these features [14] and make an intensive use of the available resources for MPI applications. MPICH-G2 introduced the concept of colors to describe the available topology. It is limited to at most four levels, that MPICH-G2 calls: 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. These approaches expose the *physical* topology for the application, which has to adapt by itself to the topology: this is the major difference with our approach. Practical experiments demonstrated that it is a difficult task to compute an efficient communication scheme without prior knowledge on the topology: the application must be written in a completely self-adaptive way.

Used along with Globus, Condor-G uses a technique called *gliding-in* [21] to run Condor jobs on a pool of nodes spanning several administrative domains. This way, a pool of Condor machines is made of the aggregation of those remote resources, the personal matchmaker and the user's Condor-G agent. This technique can be a solution for executing master-worker applications on a grid, but most non grid-specific applications are written in MPI and cannot be executed with Condor. Moreover, global operations like broadcasts and reductions cannot be done with Condor.

Collective operations have been studied widely and extensively in the last decades. However, as pointed out in [20] proposed strategies are optimal in homogeneous environments, and most often with a power-of-two number of processes. Their performance are drastically harmed in the heterogeneous, general case of number of nodes.

Topology-discovery features in Globus have been used to implement a topology-aware hierarchical broadcast algorithm in MPICH-G2 [13]. However, complex applications require a larger diversity of collective operations, including reductions, barrier, and sometimes all-to-all communications.

Grid-MPI [18] provides some optimized collective operations. The AllReduce algorithm is based on the works presented in [20]. The broadcast algorithm is based on [2]. 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, 1) this is not always a valid assumption and 2) cluster of clusters grid have a large diversity of topology and results presented in [18] only concern 2 clusters.

Previous studies on hierarchical collective operations like [6] create a new algorithm for the whole operation. Our approach tries to make use of legacy algorithms whenever possible, i.e, in homogeneous sub-sets of the system (*e.g.*, a cluster). MagPIe [15] is an extension of MPICH for aggregations of clusters. MagPIe considers as a cluster any single parallel machine, which can be a network of workstations, SMPs or MPPs. It provides a set of collective operations based on a two-level hierarchy, using a flat tree for all the inter-cluster communications. This requirement strongly limits the scope of hardware configurations. Moreover, a flat tree may not always be the most efficient algorithm for upper-level communications.

3 Architecture

In this section we present how application developers can program their applications in order to make them fit to grids. We assume two kinds of topologies: the virtual topology, seen by the programmer, and the physical topology, returned by the scheduler. The virtual topology connects MPI processes or groups of processes in a hierarchical structure; the physical topology connects resources (core, CPU, cluster, MPP...) following a hierarchical structure. We consider that a physical topology is compatible with a virtual topology if the general structure of both topologies are matching and if the physical topology is not more scattered than the virtual topology (the physical topology preserves the geographical locality of inter-process communications).

For example, if the developer requested three groups for tightly coupled processes, the scheduler can map them on three clusters, or two clusters only: both physical topologies meet the requirements of the virtual topology, since the geographical locality is preserved.

We assume that the programmer designed the virtual topology without considering a specific physical topology. However, when the programmer developed an application, he can define requirements on the physical topology through parametrization of the virtual topology. Parameters in the virtual topology are link bandwidth, link latency, available memory... (these requirements are optional). The requirements are provided to the grid meta-scheduler, that tries to allocate nodes on a physical topology matching these requirements; the allocation algorithm and a topology aware Grid meta-scheduler are described in details in [17] and are not the object of this paper.

Besides, we have developped new collective operations in the MPI library that adapt themselves to the physical topology returned by the scheduler. If we consider the aforementioned example, global collective operations will be optimized for two subsets of processes instead of three subsets of processes as required by the virtual topology. The current version of the adaptation algorithm assumes that geographical locality always reduces the collective communication execution time.

Our collective operations use the best implementation of collective operations available for every computing resource in the hierarchical structure. Compared to collective operations for homogeneous environments discussed in Section 2, our collective operations adapt themselves to the physical topology.

To submit and execute a topology-aware application, the developer writes his application and describes processes or process groups and communications in a companion file called *jobProfile*. The jobProfile is submitted to the scheduler, that provides the

```
Vanilla Ray2mesh:
Broadcasts
if I_am_master:
   while( chunk )
      distribute among workers
      receive results from workers
else /* worker */
   upon receive chunk:
      calculate ray tracing
      send results to the master
endif
Broadcast
AllToAll
Output local result
```

```
Hierarchical Ray2mesh:
Broadcasts
if I_am_central_master:
   while( chunk )
      distribute among bosses
      receive results from bosses
else
   if I_am_a_boss:
      upon receive chunk:
       while( chunk )
         distribute among workers
         receive results from workers
       send results to the central mas-
            ter or my upper-level boss
   else /* worker */
      upon receive chunk:
         calculate ray tracing
         send results to the boss
   endif
endif
Broadcast
AllToAll
Output local result
```

Figure 1: Ray2mesh, vanilla and hierarchical code versions

list of allocated machines to the launcher. The application is deployed and started on this set of machines. The MPI runtime environment obtains the physical and virtual topologies and transmits them to the MPI library (for collectives communications) and the application in order to identify the location of every MPI process in the virtual topology.

The following three subsections explain in more details how each step of the adaptation are done. Subsection 3.1 describes a specifically adapted application, Subsection 3.2 describes how the topology is adapted to the application's requirements in terms of resources and communications, and Subsection 3.3 describes a set of collective operations designed to fit on the physical topology and knowledge about proximity between processes.

3.1 Grid-enabled application

The master-worker approach is used for a very wide range of parallel applications. Its major drawback is the single point of stress (master) creating a bottleneck. We consider the class of master-worker applications where parallel computations are done from one or several large partitionnable data sets, initially located on the central master. Partitions of the data set(s), that we call ”chunks” are distributed to the workers during the execution, following a scheduling algorithm.

For these applications, after computing a chunk, a worker sends its result to the master and waits for a new chunk. Data prefetch could be used as an optimization to overlap communication and computation in order to reduce worker idle time [3]. However this approach requires strong modifications of the application, for both master and worker code and compromises the utilisation of external libraries in the application.

In a hierarchical communication and computation pattern, we introduce local masters in the virtual topology that can be used to relay data from the central master to the workers, and results from the workers to the central master. In the following, we call such a local master a *boss*. Bosses must be used at every intermediate level of the topology. A boss receives data from its upper-level boss, and sends it down to its lower-level boss or worker. Bosses are used in the virtual topology to reduce the number of cross-level communications and to foster locality of communications.

We have applied our hierarchical execution approach to Ray2mesh [12], 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). In the following, we consider the master-worker implementation of Ray2mesh.

The execution of Ray2mesh can be split up into three phases (see Figure 1). The first one consists of successive collective operations to distribute information to the computation nodes. The second phase is the master-worker computation itself. The third phase is made of collective operations to give information from all workers to all others, before they can output their part of the final result.

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 typically hierarchical topology of the grid.

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 load-balancing, not only among the workers, but also among the bosses. This property allows suiting to different sizes of clusters and different computation speeds. Moreover, it allows each boss handling fewer data requests than in an organization with a unique master.

3.2 Hardware resources and application matching

The communications of an application follow a certain pattern, which involve some requirements to be fulfilled by the physical topology. For example, tightly-coupled processes will require low-latency network links, whereas some processes that do not communicate often with each other but need to transfer large amounts of data will have bandwidth requirements. Those requirements can be described in a *JobProfile*. The jobProfile is submitted to the grid scheduler, that tries to allocate resources with respect to the requirements by mapping the requested virtual topology on available resources whose characteristics match as tightly as possible the ones requested in the JobProfile.

The JobProfile describes the process groups involved in the computation, in particular by specifying the number of processes in each group and requirements on inter- and intra-cluster communication performances. Some parameters are left blank, and filled by the meta-scheduler with the characteristics of the obtained mapping.

The *groupId* defined in the jobProfile will be passed to the application at runtime, along with the virtual topology of the resources that were allocated to the job. Using *groupIds*, it is simple to determine during the initialization of the application which group a given process belongs to, and which processes belong to a given group. The virtual topology description is passed like it was done for MPICH-G2 (*cf* Section 2), using an array of colors. Basically, two processes having the same color at a same hierarchy depth belong to the same group. In MPICH-G2, the depth of a hierarchy is limited to four. Our virtual topologies does not have this limitation.

3.3 Adapted collective operations

Collective operations are one of the major features of MPI. A study conducted at the Stuttgart High-Performance Computing Center [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.

As in other Grid-MPI work (*cf* Section 2), our MPI for grids features collective communication patterns adapted to the physical topology in order to optimize them. In the following paragraphs, we describe which collective operations have been modified for topology-awareness and how they have been modified.

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.

The broadcast has been modified for exploiting the hierarchy of the physical topology. The root of the broadcast, if it belongs to the top-level master communicator, broadcasts the message along this top-level communicator. Otherwise,the root process sends the message to a top-level process which does exactly the same thing afterwards. Each process then broadcasts the message along its “sub-masters” communicator, until the lowest-level nodes are reached.

MPI_Reduce Using associativity of the operator in the Reduce operation, it can be made hierarchical as follows: each lowest level cluster performs 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 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_Allgather aggregates data and makes the resulting buffer available on all the nodes. It can be done in a hierarchical fashion by successive *Allgather* operations from the bottom to the top of the hierarchy, followed by a hierarchical *Bcast* to propagate the resulting buffer. MPI_Barrier is similar to an MPI_Allgather without propagating any data.

4 Experimental Evaluation

We modified the runtime environment and the MPI library of the QosCosGrid Open MPI implementation presented in [8] to expose the virtual topology to the application. We also implemented the collective operations described in Section 3.3 using MPI functions.

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 [5] 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. In the last section, we present a graph showing the respective contribution of the hierachical programming and topology aware collective operations on the application performance.

4.1 Experimental Platform

Grid'5000 is a dedicated reconfigurable and controllable experimental platform featuring 13 clusters, each with 58 to 342 PCs, inter-connected through Renater (the French Educational and Research wide area Network). It gathers roughly 5,000 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-node cluster at Bordeaux, the 312-node cluster at Orsay, a 99-node 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.

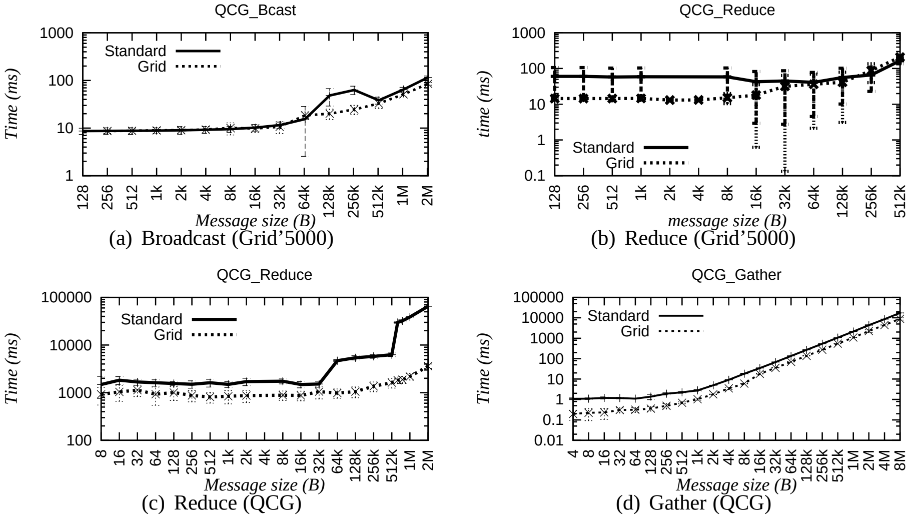
All the nodes were booted under linux 2.6.18.3 on Grid'5000 and 2.6.22 on the QCG cluster. The tests and benchmarks are compiled with GCC-4.0.3 (with flag -O3). All tests are run in dedicated mode.

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 communications have a throughput of 89.61 Mb/s and a latency of 0.1 ms.

4.2 Collective operations

We ran collective operation benchmarks on 32 nodes across two clusters in Orsay and Rennes (figures 2a-b). A configuration with two clusters is an extreme situation to evaluate our collective communications: a small and constant number of inter-cluster messages are sent by topology-aware communications, whereas $O(log(p))$ (where p is the total number of nodes) inter-cluster messages are sent by standard collective operations.

We also conducted some experiments on the QCG cluster with 8 processes mapped on each machine. Although this mapping oversubscribes the nodes (8 processes for 2 available slots), our benchmarks are not CPU-bound, and this configuration enhances



(a) Broadcast (Grid'5000)

(b) Reduce (Grid'5000)

(c) Reduce (QCG)

(d) Gather (QCG)

Figure 2: Comparison between standard and grid-enabled collective operations on a grid.

the stress on the network interface. Measurements with a profiling tool validated the very low CPU usage during our benchmark runs.

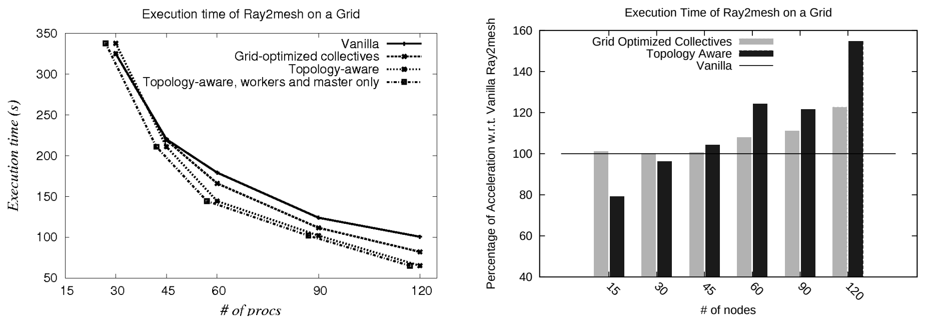
We used the same measurement method as described in [14], using the barrier described in Section 3.3 to synchronize time measurements.

Since we implemented our hierarchical collective operations in MPI, some pre-treatment of the buffers may be useful. Messages are pre-cut and sent chunk after chunk. Then it is possible to pipeline the successive stages of the hierarchical operation. It appeared to be particularly useful when shared-memory communications were involved, allowing fair system bus sharing.

Figures 2(a) and 2(b) picture comparisons between standard and hierarchical MPI_Bcast and MPI_Reduce on Grid'5000. Message pre-cutting appeared to be useful for MPI_Bcast, whereas it was useless for MPI_Reduce, since big messages are already split by the algorithm implemented in Open MPI.

One can see that, as expected, hierarchical MPI_Bcast (Figure 2(a)) always performs better than the standard implementation. Moreover, pre-cutting and pipelining permits to avoid the performance step around the eager/rendez-vous mode transition.

When messages are large regarding communicator size, MPI_Reduce (Figure 2(b)) in Open MPI is implemented using a pipeline mechanism. This mechanism allows communication costs to be 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, hierarchical communications are not an appropriate approach, and a single flat pipeline performs better.



(a) Scalability of Ray2mesh on a grid

(b) Relative acceleration of Ray2mesh, with respect to the vanilla implementation

Figure 3: Comparison of vanilla Ray2mesh with vanilla Ray2mesh using optimized collective communications, and fully topology-aware Ray2mesh

Figures 2(c) and 2(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 inter-machine 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 2(c)).

4.3 Adapted application

The execution phases of Ray2mesh are presented in Section 3.1. It is made of 3 phases: two collective 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 3(a) presents the scalability of Ray2mesh under three configurations: standard (vanilla), using grid-adapted collective operations, and using a hierarchical master-worker pattern and grid-adapted 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. The fourth line represents the values of the last configuration, measured with the same number of computing elements as in the first configuration, thus removing the local boss in the process count.

First of all, Ray2mesh scales remarkably well, even when some processes are located on a remote cluster. 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 3(a). 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 is that several processes are used to distribute the data (the bosses) instead of only one. For example, with 16 processes distributed on three clusters, 15 processes will actually work for the computation in a single-master master-worker application, whereas only 12 of them will contribute to the computation on a multi-level (two-level) master-worker application. A dynamic adaptation of the topology according to the number of involved node would select the ”non hierarchical” version for small numbers of nodes and would select the hierarchical version when the number of nodes exceeds 30.

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 3(a), the three implementations show the same performance for a small number of processes, and the grid-enabled implementations are more scalable. The performance gain for 180 processes is 35% by adding only 3 dedicated nodes working exclusively as bosses.

The relative acceleration with respect to the vanilla implementation is represented Figure 3(b). We can see that the application speed is never harmed by optimized collective operations and performs better on large scale, and a topology-aware application is necessary to get a better speedup for large-scale application.

5 Conclusion

In this paper, we proposed a new topology-aware approach to port complex MPI applications on grid through a methodology to use MPI programming techniques on grids. First we described a method to adapt master-worker patterns to grids’ hierarchical topology. 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. Then we proposed a way to describe the communication patterns implemented in the application in order to match the application’s requirements with the allocated physical topology. In the last part we presented a set of efficient collective operations that organize their communications with respect to the physical topology in order to minimize the number of high-latency communications.

Experiments showed the benefits of each part of this approach and their limitations. In particular, 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 presented an extension of the runtime environment of an MPI implementation targeting institutional grids to provide topology information to the application. These features have been implemented in an MPI library for grids.

Acknowledgements. Experiments presented in this paper were carried out using the Grid’5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https:// www.grid5000.fr), and founded by the QosCosGrid European project (grant number: FP6-2005-IST-5 033883).

References

[1] E. I. Atanassov, T. V. Gurov, A. Karaivanova, and M. Nedjalkov. Monte carlo grid application for electron transport. In *Proc. of ICCS*, volume 3993 of *LNCS*, pages 616–623. Springer, 2006.

[2] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. van de Geijn, and J. Watts. Building a high-performance collective communication library. In *Proc. of SC'94*, pages 107–116. IEEE, 1994.

[3] S. Boutammine, D. Millot, and C. Parrot. An adaptive scheduling method for grid computing. In *Proc. of Euro-Par'06*, volume 4128 of *LNCS*, pages 188–197. Springer, 2006.

[4] S. Branford, C. Sahin, A. Thandavan, C. Weihrauch, V. N. Alexandrov, and I. T. Dimov. Monte carlo methods for matrix computations on the grid. *Future Gener. Comput. Syst.*, 24(6):605–612, 2008.

[5] F. Cappello, E. Caron, and M. Dayde *et al.* Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proc. The 6th Intl. Workshop on Grid Computing*, pages 99–106, 2005.

[6] F. Cappello, P. Fraigniaud, B. Mans, and A. L. Rosenberg. HiHCoHP: Toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors. In *Proc. of IPDPS*, page 42. IEEE, 2001.

[7] M. Charlot, G. De Fabritis, A.L. Garcia de Lomana, A. Gomez-Garrido, and D. Groen et al. The QosCosGrid project. In *Ibergrid 2007 conference, Centro de Supercomputacion de Galicia*, 2007.

[8] C. Coti, T. Herault, S. Peyronnet, A. Rezmerita, and F. Cappello. Grid services for MPI. In *Proc. of CCGRID*, pages 417–424. IEEE, 2008.

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

[10] E. Gabriel, M. M. Resch, T. Beisel, and R. Keller. Distributed computing in a heterogeneous computing environment. In *Proc. of EuroPVM/MPI*, volume 1497 of *LNCS*, pages 180–187. Springer, 1998.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[12] M. Grunberg, S. Genaud, and C. Mongenet. Parallel seismic ray tracing in a global earth model. In *Proc. of PDPTA*, volume 3, pages 1151–1157. CSREA Press, 2002.

[13] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *Proc. of SPDP*, pages 377–386. IEEE, 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] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. M*AGPI*E: MPI's collective communication operations for clustered wide area systems. In *Proc. of PPoPP*, volume 34.8 of *ACM Sigplan*, pages 131–140. ACM Press, 1999.

[16] V. Kravtsov, D. Carmeli, A. Schuster, B. Yoshpa, M. Silberstein, and W. Dubitzky. Quasi-opportunistic supercomputing in grids, hot topic paper. In *Proc. of HPDC*, 2007.

[17] V. Kravtsov, M. Swain, U. Dubin, W. Dubitzky, and A. Schuster. A fast and efficient algorithm for topology-aware coallocation. In *Proc. of ICCS*, volume 5101 of *LNCS*, pages 274–283, 2008.

[18] M. Matsuda, T. Kudoh, Y. Kodama, R. Takano, and Y. Ishikawa. TCP adaptation for MPI on long-and-fat networks. In *Proc. of CLUSTER*, pages 1–10. IEEE, 2005.

[19] P. Nascimento, C. Sena, J. da Silva, D. Vianna, C. Boeres, and V. Rebello. Managing the execution of large scale mpi applications on computational grids. *Proc. of SBAC-PAD*, pages 69–76, 2005.

[20] R. Rabenseifner. Optimization of collective reduction operations. In *Proc. of ICCS*, volume 3036 of *LNCS*, pages 1–9. Springer, 2004.

[21] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.