Future Generation Computer Systems 27 (2011) 357–369

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

QCG-OMPI: MPI applications on grids

Emmanuel Agullo [c], Camille Coti [b,*], Thomas Herault [d,e,b], Julien Langou [f], Sylvain Peyronnet [d], Ala Rezmerita [d], Franck Cappello [b,g], Jack Dongarra [a]

[a] Department of Electrical Engineering and Computer Science, University of Tennessee, 1122 Volunteer Blvd, Claxton Building, Knoxville, TN 37996-3450, USA
[b] LRI/INRIA Saclay-Île de France, Orsay, F-91893, France
[c] LaBRI/INRIA Bordeaux-Sud-Ouest, Talence, F-33405, France
[d] Univ Paris-Sud, UMR8623, LRI, Orsay, F-91405, France
[e] CNRS, Orsay, F-91405, France
[f] Department of Mathematical and Statistical Sciences, University of Colorado Denver, Campus Box 170, P.O. Box 173364, Denver, CO 80217-3364, USA
[g] UIUC, T. M. Siebel Center for Computer Science, Urbana, IL 61801-2302, USA

ARTICLE INFO

Article history:
Received 5 March 2010
Received in revised form
9 November 2010
Accepted 12 November 2010
Available online 26 November 2010

Keywords:
Grid computing
MPI
Topology-aware middleware
Applications
Dense linear algebra
Collective communications
Master–worker

ABSTRACT

Computational grids present promising computational and storage capacities. They can be made by punctual aggregation of smaller resources (*i.e.*, clusters) to obtain a large-scale supercomputer. Running general applications is challenging for several reasons. The first one is inter-process communication: processes running on different clusters must be able to communicate with one another in spite of security equipments such as firewalls and NATs. Another problem raised by grids for communication-intensive parallel application is caused by the heterogeneity of the available networks that interconnect processes with one another. In this paper we present how QCG-OMPI can execute efficient parallel applications on computational grids. We first present an MPI programming, communication and execution middleware called QCG-OMPI. We then present how applications can make use of the capabilities of QCG-OMPI by presenting two typical, parallel applications: a geophysics application combining collective operations and a master–worker scheme, and a linear algebra application.

1. Introduction

Computational grids present promising computational and storage capacities. They can be made by punctual aggregation of smaller resources (*i.e.*, clusters) to obtain a large-scale supercomputer (*i.e.*, the resulting grid). Many large-scale scientific problems have been successfully solved, thanks to the use of computational grids in biology (protein folding [1]), medicine (cure muscular dystrophy [2]), financial modeling, earthquake simulation, and climate/weather modeling [3]. Such applications are very specific and are particularly well adapted to computing on a large-scale, loosely-coupled platform and have been implemented without any direct connection between computing processes. On the other hand, practical experience has shown that applications involving point-to-point communication between processes perform poorly on grids [4].

Running general applications is challenging for several reasons. The first one is inter-process communication: processes running on different clusters must be able to communicate with one another. On the other hand, clusters must be protected from external intrusions by security equipments such as firewalls and NATs, which additionally are used to reduce the number of public IP addresses needed by clusters. To address this problem without requiring any trade off between security and connectivity, we designed QCG-OMPI[1] [5,6], an extended MPI library based on a framework providing several basic and advanced connectivity techniques aiming at firewall and NAT bypassing.

The QosCosGrid[2] [7] project aims at developing and executing parallel applications on institutional grids. A full job management and execution stack has been designed in order to support applications on grids. QosCosGrid uses QCG-OMPI as its MPI implementation.

Another problem raised by grids for communication-intensive parallel application is caused by the heterogeneity of the available networks that interconnect processes with one another. Local communication media have a lower latency and a higher bandwidth

* Corresponding author.
E-mail addresses: coti@lri.fr, camille.coti@lipn.univ-paris13.fr (C. Coti).

[1] QosCosGrid-OpenMPI, QosCosGrid standing for Quasi-Opportunistic Supercomputing in Grid environments.

[2] Quasi-Opportunistic Supercomputing for Complex Systems in Grid Environments, http://www.qoscosgrid.eu.

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.11.015

than long-distance networks by two to four orders of magnitude. Hence, applications must follow adapted communication schemes in order to avoid lower performance, higher cost communications (*i.e.*, inter-cluster communications) and, as a consequence, adapted computation schemes.

The approach commonly followed so far has consisted in discovering the available physical topology at run-time and adapt the application dynamically. Nonetheless, experience has shown that writing applications that can adapt themselves to *any* topology is a difficult problem and often refrains programmers from using such platforms. The approach used here inverts the process of matching the application's topology to the physical topology and pushes the complexity of this process to the grid meta-scheduler [8].

Contributions of this paper

In this paper we present how QCG-OMPI can execute efficient parallel applications on computational grids. Pieces of this work have been published as proceedings of several conferences [9,8, 10,6]. Their respective focus ranged from the core concepts of QCG-OMPI [6] and its design [8] to its usage for practical applications [10,9]. In the present paper, we synthesize those results into a standalone study. We also draw conclusions on this project and propose this approach for the future cloud platforms.

The paper is organized as follows. We first present previous works on MPI on the grid, adapted middleware and applications on the grid (Section 2). Section 3 presents the architecture of QCG-OMPI: its connectivity features (Section 3.1) and the method we propose here to program high performance applications for the grid (Section 3.4). Section 4 presents the communication performance obtained by QCG-OMPI for both point-to-point and collective communications, and Section 5 presents two typical applications adapted for the grid using QCG-OMPI.

2. Background

MPI [11] is the *de facto* standard for programming parallel applications. As a consequence, it is a natural choice for programming parallel applications for the grid to re-use existing parallel routines and expertise in programming scientific applications using MPI. Open MPI [12] and MPICH [13] can be cited as two of the main open-source implementations of the MPI standard.

Running MPI applications on computational grids raises two major issues. The first obstacle is a technical issue concerning communications throughout the grid. The second difficulty is the need for a programming technique to be used to obtain good performance on such platforms.

2.1. MPI middleware and the grid

The two major implementations have been designed for clusters, without considering the particularities of grids. Several MPI implementations have been designed or adapted to deal specifically with some of these particularities. PACX-MPI [14,15] uses relay daemons to forward inter-cluster communications and a local MPI implementation available on each cluster for intra-cluster communications; this approach allows to handle data representation heterogeneity by converting the payload of inter-cluster messages into a common data format, called eXternal Data Representation (XDR). It also limits the number of ports that have to be open in the firewall, and can handle Network Address Translation (NAT) if the relay daemon is located on the NAT server. However, relay daemons induce an extra hop in inter-cluster communications, and can quickly become a bottleneck.

MPICH-G2 [16,17] is based on the MPICH [13] library and uses the Globus Toolkit (GT) [18] to schedule, deploy and coordinate a job across the grid. It does not include any firewall nor NAT bypassing mechanism, although it can be set up to use a specific range of ports that correspond to ports that are left open by the firewall. Another drawback of MPICH-G2 is its complex architecture and its extensive use of external services (from the Globus Toolkit) during the start-up phase, which is, consequently, responsible for significant overhead in the job's life cycle. MGF [19] is based on MPICH-G2 and implements relay proxies and a delegation mechanism that are similar to PACX-MPI's techniques for transparent inter-cluster communications between clusters using private addressing and efficient collective operations.

GridMPI[3] uses the Interoperable MPI (IMPI) [20] standard to interconnect multiple instances of MPI. It is based on the YAMPII[4] MPI library. It can also use relay daemons to pass through firewalls with only one open port, but requires global IP addressing. Moreover, it cannot handle non-trivial heterogeneity of data representation, like different floating point and 32/64 bits support, whereas Open MPI allows heterogeneous 32/64 bits representation, and can handle more complex shifting between different architectures.

2.2. Collective operations on the grid

Collective operations have been studied widely and extensively over the last decades. According to a study conducted on an intensively-used Cray T3E 900-512 supercomputer [21], almost every MPI program use collective operations whereas only 18.9% of the CPU time is spent on programs that use point-to-point communications. Besides, this study shows that 58.9% of the time spent in MPI routines is spent doing collective operations.

Collective communications are most often based of tree-like topologies, like Fibonacci [22] or binary trees. Optimizations like halving and doubling increase the available bandwidth [23]. However, these algorithms were proved optimal in homogeneous environments, and suffer a high overhead in systems where not all the networks links have the same performance. If one message transmission takes longer than the other ones, it may cause a slow down in the whole system, as a side-effect of the algorithm.

A theoretical study of a broadcast algorithm for hierarchical architectures as been done by Cappello et al. in [24]. They propose a top-to-bottom hierarchical approach to broadcast a message within groups placed at the same level of hierarchy. One effect of this approach is that the number of upper-level, expensive messages is reduced to a minimum value of $O(1)$. We propose here to generalize this approach for any collective operation.

2.3. Programming MPI applications for the grid

Programming applications for the grid requires some adaptation to the underlying topology of the computing resources. Practical experiments using the Amazon EC2 Cloud [4] and as part of the GrADS[5] project [25] showed that parallel applications designed for clusters and executed directly (without any adaptation) on grids suffer from the usage of remote resources and are dramatically slowed down when using a system which is not a single cluster. MPICH-G2 introduced the concept of colors to describe the available topology with four levels of hierarchy called: WAN, LAN, system area and, if available, vendor MPI. However, one can expect finer-grain topology information and more flexibility for large-scale grid systems. These approaches expose the physical topology to the application, which has to dynamically adapt itself to the

[3] GridMPI official webpage: http://www.gridmpi.org.

[4] YAMPII official web page: http://www.il.is.s.u-tokyo.ac.jp/yampii.

[5] Software Support for High-Level Grid Application Development http://www.hipersoft.rice.edu/grads/.

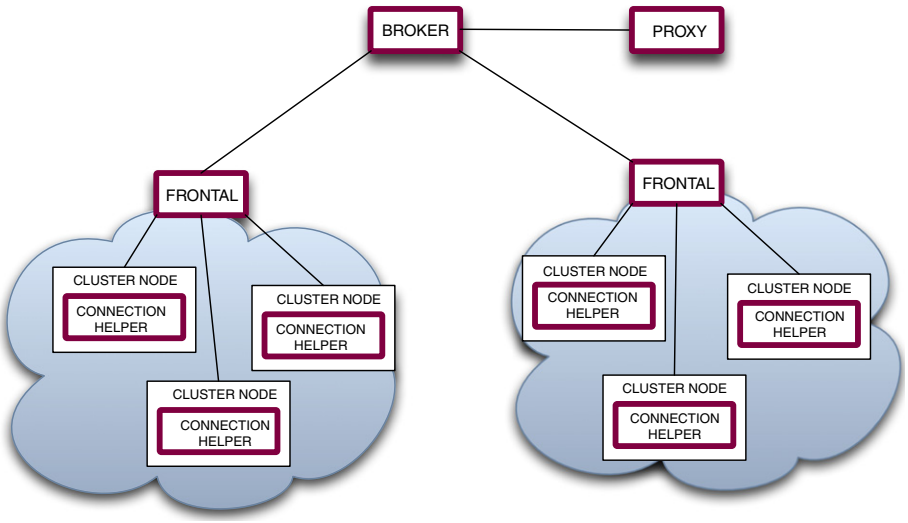


Fig. 1. Architecture of the grid infrastructure of QCG-OMPI.

available topology. A self-adaptive master–worker algorithm [26] is an attempt to adapt the application at run-time.

Practical experiments have shown that it is easier to write an application with an *a priori* knowledge of the topology it will be executed on. For example, [27] is based on the MPI routine `MPI_Scatterv` to scatter chunks of data of different size among computing resources. Following the same static approach, [28] presents a sorting algorithm that balances the amount of work to do regarding what each resource can handle and what their computing speed is.

3. Architecture

In this section we present the architecture of QCG-OMPI, an adaptation of Open MPI [12] specifically designed for computational grids following the "cluster of clusters" design. In 3.1 we present how QCG-OMPI deals with communications throughout the grid, in spite of security equipments supposed to limit connectivity. Connectivity techniques are presented in Section 3.2. Section 3.3 presents how collective operations can be adapted to the topology using this approach. In 3.4 we present a method for programming efficient parallel applications on grids with respect to the specific requirements such as topology.

Key ideas of QCG-OMPI. QCG-OMPI features connectivity techniques that permit *direct point-to-point communications* between processes across the grid, without requiring any specific configuration of the security and routing equipments. It provides a set of grid-optimized, *hierarchical collective communications* to allow applications to communicate efficiently and following adapted patterns. QCG-OMPI also provides an extension of the MPI standard and collaborates with a grid-enabled scheduler to allow the development and execution of *grid-enabled, hierarchical applications* that avoid high latency communications by using appropriate communication and computation patterns.

3.1. Communication middleware

One of the first problems that appear while trying to run an MPI application over a computational grid is the connectivity of the different nodes not belonging to the same administrative domain. The presence of security equipments like firewalls and Network Addresses Translators (NAT) – supposed to protect the domain from external attacks – limits the connectivity of parallel applications.

In order to solve this connectivity limitation QCG-OMPI features an extent-ion of Open MPI's run-time environment and library with a set of grid services providing the communication library with advanced connectivity capabilities. These services abstract the connectivity service and provide the transparent communication establishment for the MPI applications throughout a computational grid. Fig. 1 presents a global picture of the architecture, across two administrative domains.

Domains must be reachable from one another, at least through gateway nodes, on which we run what we call a *frontal* service (or component). In addition to services running on each administrative domain, we use two services that can be hosted by any administrative domain, namely the *broker* and one or multiple *proxies*. The *broker* service must be reachable from any *frontal* component and the *proxy* service must be reachable from any node. On every node of each administrative domain, we also use a *connection-helper* service that is connected to the *frontal* service.

These services have been implemented using the SOAP remote procedure call protocol in order to be portable and interoperable across hardware and software architectures. We used the lightweight grid service engine gSOAP [29]. The following section describes their main characteristics. They have been described more thoroughly in [6].

3.1.1. The broker

The *broker* service is a centralized service running on only one machine in the grid and accepting incoming connections from *frontal* components. The brokering service provides a way to communicate between nodes located in the same administrative domain and between nodes that would not be able to establish connections with one another otherwise, because a NAT and/or a firewall are standing between them. To do so, the *broker* receives the local cluster configuration from the *frontal* service. This configuration includes the connectivity restrictions (cluster is open on a port range, firewall bypassing techniques, or completely closed). In addition of collecting all the local configurations, the brokering service centralizes all processes contact information. This process contact information corresponds to the Open MPI process unique identifier called process name and the process access point (the public IP address of the node where the process is executed and the port number on which the process listens for incoming connections). The *broker*'s global contact list is filled by every Open MPI process and takes place every time an Open MPI process creates an access point.

Each time the Open MPI process wants to establish a communication with another Open MPI process, it invokes the brokering service through *connection-helper* and *frontal* services. Cross checking all local configurations, the *broker* finds which technique is most relevant to establish the requested connection.

3.1.2. The frontal

The *frontal* service is running on a front-end machine of each cluster. It is connected to the brokering service and it accepts connections from *connection-helper* components. The *broker* sends answers to the *frontal*, that saves it in its local memory. When another process from this cluster asks for this information again, the *frontal* can answer directly without asking the *broker*. This hierarchical communication pattern provides scalability to the architecture, with a small cost on latency to connect to the *broker*.

3.1.3. The connection-helper

The *connection-helper* service runs on every node. The goal of this component is to help any Open MPI process to establish the connection with another process by relaying messages between Open MPI process and the *frontal* service before the connection is established.

3.1.4. The proxy

We assume that every QCG-OMPI process and every cluster node can access this service directly. The *proxy* service is a service running on one or many machines in the grid and accepting incoming requests from the *broker* service. The goal of this service is to relay messages between nodes that are not able to establish a direct connection with each other.

Multiple *proxy* processes can be launched independently on the grid (as long as they are accessible from any point in the grid). The brokering service balances the load between all the available *proxy* services using a simple round-robin heuristic.

3.2. Connectivity

In order to solve the connectivity problems we implemented several basics techniques (direct connection, port range technique, relaying) and one advanced method (*Traversing TCP* [30]) that allow direct connections through firewalls without being a threat to the security of the network.

3.2.1. Direct connection

The brokering service stores all local configurations and all nodes access point. When the Open MPI process on node A is willing to communicate with the Open MPI process on node B, it invokes the brokering service through *connection-helper* and *frontal* services. As result of this new connection request, if the node A is able to contact directly node B, the brokering service returns node B's contact point to node A, allowing node A to open a direct connection to node B.

3.2.2. Port range technique

In case an open port range for incoming connections is available on the firewall of the grid site, the run-time environment ensures that all sockets used for incoming connections in this site are bound to a port in this range. The Open MPI process is informed about the known port range through Modular Component Architecture (MCA) parameters.

3.2.3. Relaying

The most reliable connectivity method implemented in QCG-OMPI is relaying. This technique is designed for the nodes that reside in different administrative domains, and that are such that their respective Firewall/NATs prevent either node from directly initiating a connection to the other. Relaying always works as long as both cluster nodes can connect to the *proxy* server. Instead of attempting a direct connection, the two nodes can simply use the *proxy* server to relay messages between them.

3.2.4. Traversing TCP

Traversing TCP [30] is derived from the TCP protocol and it works with firewalls that are not running stateful packet inspection. It essentially consists in transporting, using the *broker*, the initiating TCP packet (SYN) blocked by the firewalls or NAT on the server side and injecting the packet in the server's IP stack.

During the connection establishment the client application's SYN packet is intercepted by the *connection-helper* and forwarded to the *connection-helper* on the server side. Once the packet received, the server *connection-helper* injects the SYN packet to the IP stack. The server application replies, to this SYN packet, with a SYN/ACK packet. The initialization of the TCP connection ends with an ACK packet from the client to the server: the TCP connection is established.

Once the connection is established, the communication can continue between the two processes following classical TCP operations. The communication between the two processes is direct, bypassing the *broker* and ensuring high point-to-point communication performance.

3.3. Collective operations

The basic idea for optimizing communication patterns on the grid consists in communication topologies designed such in a way that they aim at reducing high latency communications, as presented for a broadcast operation in [24]. In other words, inter-cluster communications must be avoided on clusters of clusters, and inter-machine communications must be avoided on clusters of multicores.

A naive, cluster-optimized collective communication algorithm like binomial tree used on a grid will send $O(\log_2 C)$ inter-cluster messages and $O(\log_2 \frac{P}{C})$ intra-cluster messages, if P denotes the *total* number of processes and C denotes the number of clusters. The goal is to schedule communications in order to send a minimal number of inter-cluster messages, *i.e.*, $O(1)$. For example, a broadcast operation needs to send only one inter-cluster message to make it reach a given cluster; then this message will be broadcast within this cluster using $O(\log_2 \frac{P}{C})$ intra-cluster messages.

The same approach can be generalized and followed to adapt other communication routines. The algorithms themselves have been described in [31].

3.4. Programming MPI applications for the grid

Computational grids are intrinsically hierarchical. They are built by federating more or less remote resources: multicore machines are aggregated to build clusters interconnected by high speed, low latency networks. Several clusters can be located in the same building and be interconnected by Ethernet networks. Several organizations can interconnect their clusters through the Internet. As a consequence, applications must be programmed in a hierarchical way and follow hierarchical communication and computation patterns.

3.4.1. Topology adaptation mechanism

Programming applications that can adapt themselves to any underlying physical topology at run-time is a complex task. Computation patterns must adapt themselves to the communication patterns induced by the physical topology the application is being executed on and load balancing must be done dynamically.

The QosCosGrid approach considers the problem of matching the virtual topology of the application and the physical topology

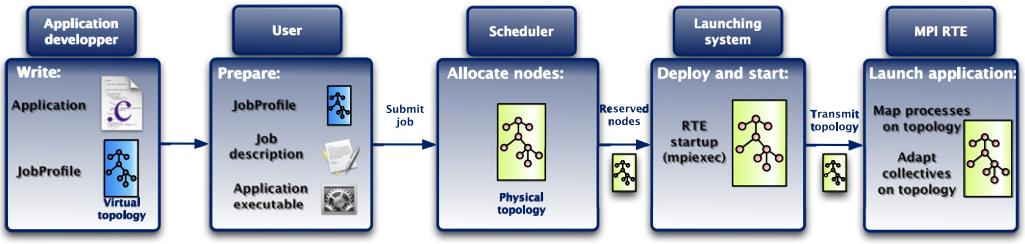


Fig. 2. Development and life cycle of a QCG-OMPI application.

of the available resources the other way around: instead of asking the application to adapt itself to any topology, the meta-scheduler allocates resources with respect to the application's virtual topology. Indeed, the complexity of executing applications that use a given virtual topology on an appropriate set of resources is pushed to the meta-scheduler. The development chain and life cycle of an application executed on the QosCosGrid system is represented in Fig. 2.

Virtual topology. The application's virtual topology is described in a companion file called *JobProfile*, which contains a description of process groups, hardware resource requirements on the computing resources used to execute these processes on such as CPU speed and available memory, as well as requirements on the interconnection characteristics between and within these process groups. An excerpt of a JobProfile is shown in Fig. 3. It is strongly inspired by the network description language presented by Lacour et al. in [32].

Job submission and scheduling. This companion file is passed to the grid meta-scheduler when the application is submitted. The grid meta-scheduler has enough knowledge to allocate resources, if available, with respect to the virtual topology described in the JobProfile. It therefore binds a physical topology accordingly.

The scheduling mechanism used by QosCosGrid has been described in [33]. The *resource offer*, *i.e.*, the resources available in the system and their properties (CPU frequency, interconnection network ...) are described in a *Resource Topology Graph* (RTG). The scheduling framework translates the request, expressed in the JobProfile, into a *request RTG*. Based on these two inputs, the scheduler tries to allocate resources in accordance with the user's request.

Job deployment and execution. QosCosGrid [34,35] uses QCG-OMPI as its MPI implementation. QCG-OMPI features the necessary additions to the MPI language to present the topology chosen by the QosCosGrid meta-scheduler [36] to the application.

4. Communication performance

This section presents the performance obtained by the QCG-OMPI communication library. First, we present the platform used to run these experiments in Section 4.1. Section 4.2 compares the performance obtained by the connectivity techniques featured by QCG-OMPI. Section 4.3 presents a set of grid-enabled collective communication algorithms and the performance obtained on two typical configurations: a cluster of multicore nodes and a cluster of clusters.

4.1. Experimental platform

The performance evaluations presented in the section and in the following one were conducted on two testbeds. The first platform we used is the QCG cluster, gathering 4 multicore-based nodes with dual-core Intel Pentium D (2.8 GHz/2 × 1 MB L2 cache) processors interconnected by a 100 MB Ethernet network. The second testbed we used is Grid'5000 [37], a dedicated reconfigurable and controllable experimental platform featuring 13 clusters, each with 58 to 342 PCs, interconnected through Renater (the French Educational and Research wide area Network). It gathers roughly

```
<grmsJob appId="ray2mesh-job">
  <resourceTemplates>
    <!– Define resources and network requirements –>
    <networkResourceTemplate templateId="net-high">
      <networkParameter name="bandwidth">
        <min>5120</min>
      </networkParameter>
    </networkResourceTemplate>
    <networkResourceTemplate templateId="net-local">
      <networkParameter name="latency">
        <max>5</max>
      </networkParameter>
    </networkResourceTemplate>
  </resourceTemplates>

  <task taskId="ray2mesh-task1">
    <resourcesDescription>
      <topology>
        <group groupId="central_master">
          <processes>
            <!– Describe each group in terms of number of
                processes and resource requirements –>
          </processes>
          <processesConnection>
            <!– Define network requirements in groups and
                between groups –>
            <networkResource>
              <templateIdReference templateId="net-local" />
            </networkResource>
            <groupConnection endpointGroupId="group1">
              <networkResource>
                <templateIdReference templateId="net-high" />
              </networkResource>
            </groupConnection>
          </processesConnection>

        </group>
      </topology>
    </resourcesDescription>

    <execution type="open_mpi">
      <!– Give path to the application, provide command-line
          arguments, path to input data files and
          standard input/outputs –>
    </execution>
  </task>
</grmsJob>
```

Fig. 3. Excerpt of a JobProfile defining one process group and network resource requirements (the other groups are not represented in this example). The processes in the group "*central_master*" must be interconnected with one another according to the "*net-local*" latency requirement, and with the processes of the group "*group*1" (not shown here) under the "*net-high*" bandwidth requirement.

Table 1
Latency comparison for all techniques (in seconds).

Times in seconds	MPICH-G2	Open-MPI	QCG-OMPI direct	QCG-OMPI proxy	QCG-OMPI traversing
Intra-cluster	0.0002	0.0001	0.0001	N/A	N/A
Orsay–Rennes	0.0104	0.0103	0.0103	0.0106	0.0103
Orsay–Bordeaux	0.0079	0.0078	0.0078	0.0084	0.0078
Bordeaux–Rennes	0.0081	0.0080	0.0080	0.0287	0.0080

Table 2
Bandwidth comparison for all QCG-OMPI techniques (in Mb/s).

Throughput in Mb/s	QCG-OMPI direct	QCG-OMPI proxy	QCG-OMPI traversing
Intra-cluster	894.39	N/A	N/A
Orsay–Bordeaux	136.08	76.40	138.18

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/1 MB L2 cache) bi-processors. This includes 5 of the 13 clusters of Grid'5000: a 93-node cluster at Bordeaux, the 312-node cluster at Orsay, a 99-node cluster at Rennes, a 56-node cluster in Sophia-Antipolis and a 60-node cluster in Toulouse. Nodes are interconnected within each cluster by a Gigabit Ethernet switch.

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 were compiled with GCC-4.0.3 (with flag -O3). All tests were run in dedicated mode: nodes were reserved thanks to a reservation system which ensures that no other user could log on them during the experiments.

4.2. Point-to-point communications

Basic communication benchmarks using NetPipe[6] provide a comparison of the techniques available in QCG-OMPI. In Table 1 we compare the communication latency obtained by QCG-OMPI, Open MPI and MPICH-G2 using three clusters located in Orsay, Rennes and Bordeaux. When a relay proxy is used by QCG-OMPI, it is located in the Orsay cluster. Intra-cluster communications always use direct connection.

As expected, all the communication libraries that establish a direct connection between two processes have the same performance. The TCP traversing technique establishes a direct connection as well and therefore achieves the same performance.

Since QCG-OMPI's proxy technique involves an extra hop through the proxy between the two communicating processes, the latency is increased regarding the location of the proxy. When at least one process is located in the same cluster as the proxy, this extra hop adds the cost of an intra-cluster communication. When none of the processes is in the same cluster as the proxy (*e.g.*, Bordeaux–Rennes) it adds the cost of an inter-cluster communication and doubles the latency.

Table 2 compares the bandwidth obtained by the different connectivity techniques provided by QCG-OMPI. When a relay proxy is used to communicate between two processes, the proxy's bandwidth is shared between the two processes. As a consequence, the available bandwidth is divided by two.

4.3. Collective operations

This section presents a set of grid-enabled collective communication algorithms and their performance on hierarchical platforms.

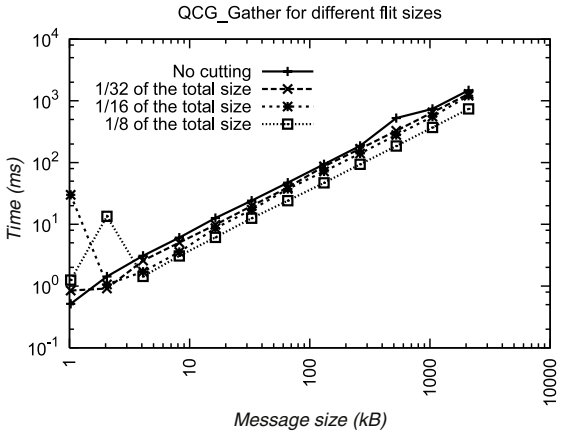


Fig. 4. Effects of message pre-cutting on QCG_Gather.

The performance showed here are those of collective operations based upon the approach described in Section 3.3. Communications are made within process groups and between process groups separately, in order to minimize the number of inter-cluster communications by letting only *one* process of each group take part of the inter-group communication. For example, for a reduction on a grid, a local reduction is made within each cluster toward one process of this cluster (root of the local operation), and a reduction is make between the roots of each cluster.

4.3.1. Implementation optimization

Our implementation of these algorithms uses an optimization inspired by Wormholes [38]: messages can, in some situations, be pre-cut into small pieces called *flits* in order to introduce a pipeline between the levels of hierarchy. Fig. 4 presents the effects of pre-cutting messages for a *gather* operation. We can see that for medium to large messages, best performance is reached when messages are cut into eight flits. When a flit has reached the following level of hierarchy, the following flit can be sent, and so on. The message can span simultaneously over several levels of hierarchy. This optimization increases the available bandwidth thanks to the pipeline thus made available. It appeared to be particularly useful when shared memory communications were involved, allowing fair system bus sharing.

4.3.2. Performance evaluation

Experimental platform. We conducted two series of experiments, each one on a platform representing a typical, hierarchical architecture. These two platforms are described in Section 4.1. We used the QCG cluster as a cluster of multicores, and Grid'5000 as a grid. Since we conducted both cluster and grid experiments, we used direct point-to-point connections between processes, in order to use the same connectivity technique in both series of experiments.

Experimental setup. The experiments we conducted on the QCG cluster present measurements in a typical configuration of a cluster of multicores. We mapped 8 processes on each node to get 32 processes in total. We deliberately chose to oversubscribe the nodes in order to stress the network (and more particularly the system bus) more intensively. Since our micro-benchmarks are communication-intensive and not computation-intensive, our experiments have not been biased by the fact that several processes are sharing the same core. Measurements with a profiling tool validated the very low CPU usage during out benchmark runs (close to 100% of the execution time was spend in MPI routines, according to the profiling tool mpiP[7]).

[6] http://www.scl.ameslab.gov/netpipe.

[7] http://mpip.sourceforge.net.

Table 3
Communication performances on QCG and Grid5000.

	Throughput in Mb/s	Latency in ms
(a) QCG		
Shared memory	3979.46	0.02
TCP	89.61	0.1
(b) Grid5000		
Intra-cluster	894.39	0.1
Orsay–Bordeaux	136.08	7.8

We conducted grid experiments on Grid'5000 using the Orsay and Bordeaux clusters and mapping 16 processes on each cluster (1 process per machine). This configuration is an extreme situation for testing our communication algorithms since it requires a minimum number of inter-cluster communications.

We followed the same experimental methodology as described in [17], using the grid-enabled barrier to synchronize time measurements. Communication performances on QCG and Grid5000 are summarized in Table 3.

Performance. We compare the performance of our grid-enabled collective communication algorithms with the standard algorithms used in Open MPI in Fig. 5. The first row of plots shows comparisons on Grid'5000 and the second one shows comparisons on the QCG cluster. As expected, the hierarchical MPI_Bcast (Fig. 5(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 with respect to the communicator size, MPI_Reduce (Fig. 5(b)) is implemented in Open MPI using a pipeline mechanism. Indeed, 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. Since hierarchy shortens the pipeline, its latency (*i.e.*, the time to load the pipeline) is smaller. Short messages are thus processed more efficiently. On the other hand, 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.

Fig. 5(c) and (d) picture comparisons between standard and hierarchical MPI_Reduce and MPI_Gather on the QCG cluster. On a cluster of multicores, 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 (Fig. 5(c)).

5. Applications

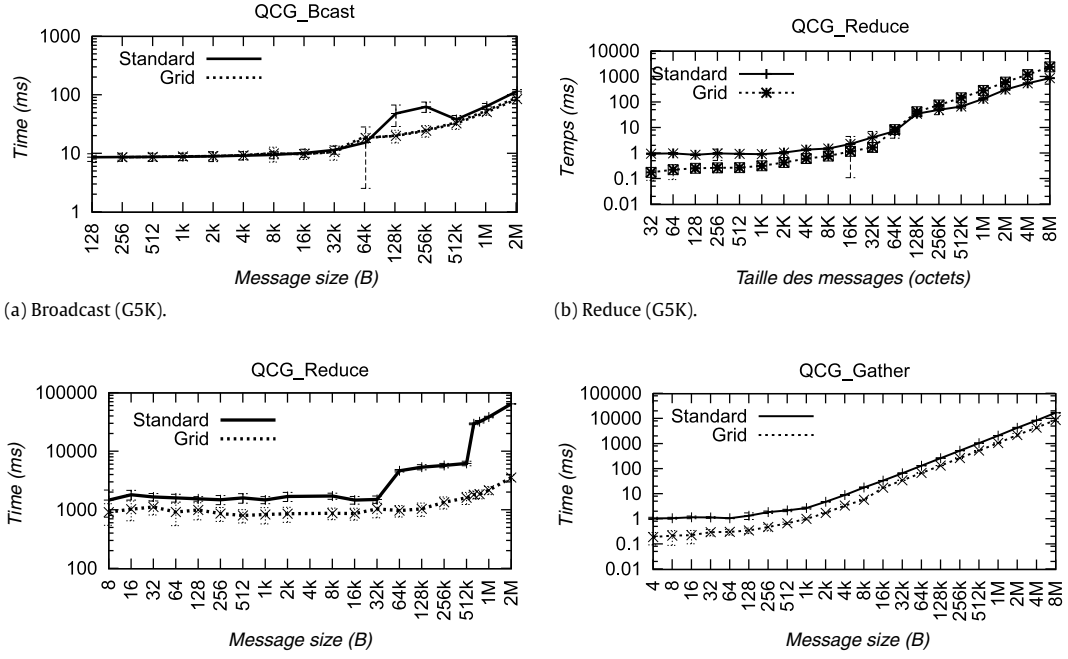
The global approach of QosCosGrid and QCG-OMPI makes possible the resolution of complex, non-trivial problems on a computing grid. In this section we present two typical applications adapted for computational grids. In Section 5.1 we present a geophysics application using collective communications and a master–worker computation scheme. In Section 5.2 we present a linear algebra application that computes an operation numerous applications are based on. More details on these applications can be found in [10,9].

5.1. Master–worker application

The master–worker communication and computation scheme is widely used for massively parallel applications, when the input data can be cut into sub-tasks (called "*chunks*") that can be computed independently from one another. It features numerous interesting properties, such as automatic load balancing. However it suffers some drawbacks when it comes to large-scale. The major one is the fact that the data is distributed from and the results are gathered by a single process (the master), creating a bottleneck.

5.1.1. Articulation with QCG-OMPI

As described in Section 3.4, we propose to follow a hierarchical approach in our data distribution and result gathering algorithm. The basics are still the same as with traditional master–worker:



(a) Broadcast (G5K).

(b) Reduce (G5K).

(c) Reduce (QCG).

(d) Gather (QCG).

Fig. 5. Comparison between standard and grid-enabled collective operations on a grid.

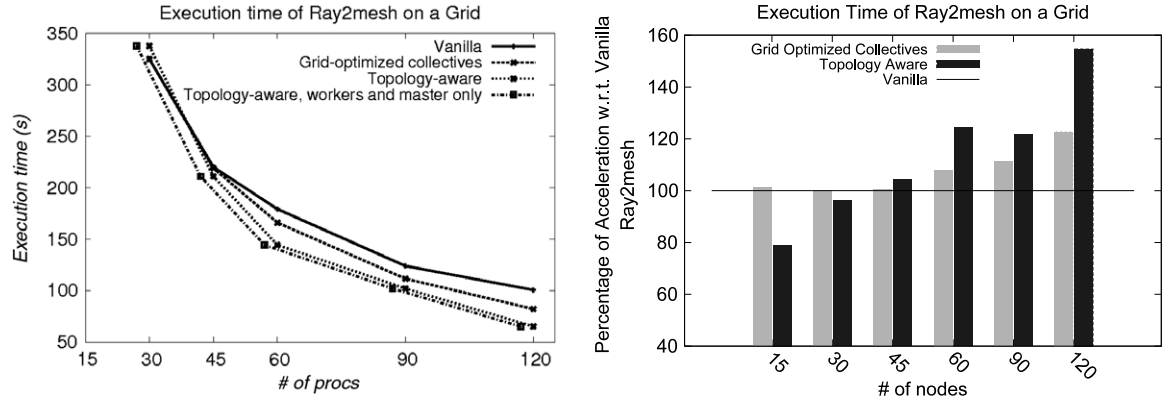
Vanilla Ray2mesh:

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

Hierarchical Ray2mesh:

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

Fig. 6. Ray2mesh, vanilla and hierarchical algorithms.



(a) Scalability of Ray2mesh on a grid.

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

Fig. 7. Comparison of vanilla Ray2mesh with vanilla Ray2mesh using optimized collective communications, and fully topology-aware Ray2mesh.

data is kept in queues, chunks are distributed and results are gathered.

In order to follow a hierarchical approach, we introduced a new kind of process called "*bosses*". Each process group has one boss; an upper-level boss is chosen among bosses within a given level of hierarchy. The top-level boss is the *central master*.

As seen by their workers or lower-level bosses, bosses act like masters: they send chunks of data and collect results. When the computation of a set of data is done they send the results to their upper-level boss to request for another big chunk of data.

Ray2mesh [39] is a geophysics application that calculates seismic rays along a given mesh containing a geographic area description, using the Snell–Descartes law in spherical geometry to propagate a wave front from a source (earthquake epicenter) to a receiver (seismograph).

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. Furthermore, the master–worker computation can suffer from a central point of contention at large-scale.

We implemented a grid-enabled version of Ray2mesh using hierarchical collective operations as described in Section 4.3 and the hierarchical master–worker algorithm described above and in Fig. 6. Each process group corresponds to a communicator: lower-level processes (*i.e.*, workers and their local boss) in a given process group share a communicator. A process group is executed on a single cluster. A head process is determined in this communicator and is referred to as the local boss. Local bosses share a communicator and a head process is determined among them, and so on until the super-master (top-level boss) is reached. These communicators are also used by collective operations to adapt themselves to the topology.

5.1.2. Performance

We evaluated the performance of Ray2mesh and our hierarchical implementation on Grid'5000 using three clusters: Orsay, Bordeaux and Rennes.

Fig. 7(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 Fig. 7(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 Fig. 7(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 in Fig. 7(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.

5.2. Linear algebra

We now show how QCG-OMPI can enable the articulation of a dense linear application with the topology of the grid in order to confine intensive communications within clusters and limit inter-cluster exchanges. A more detailed study, from which this section was extracted, can be found in [9].

5.2.1. QR factorization of tall and skinny matrices

The *QR* factorization of a general $m \times n$ real matrix A is the decomposition of that matrix into the product $A = QR$ of an $m \times m$ real orthogonal matrix Q and an $m \times n$ real upper triangular matrix R. If A is non-singular and if the diagonal elements are positive, then this decomposition is unique. There are several algorithms to perform a *QR* factorization. We focus on the computation of this product using Householder reflections (reflections about a plane) because of their backward stability. This algorithm consists of the successive application of such Householder reflections of the form $H = I - \tau v v^T$ where v is a column reflector, τ is a scaling factor and I is the identity matrix [40]. To benefit from an efficient cache usage, state-of-the-art implementations perform a *blocked* factorization [41]. ScaLAPACK [42], accumulates b elementary Householder reflectors within a *panel* (a block-column) V. The subsequent applications of these b reflectors $(H_1 H_2 \ldots H_b)$ is then applied all at once using the matrix equality $H_1 H_2 \ldots H_b = I - VTV^T$ (T is a $b \times b$ upper triangular matrix). However, within a panel, columns are factorized successively inducing a reduction after each column factorization. If P processors are used, at least $b \log_2(P)$ messages are required to perform the panel factorization.

We consider a variant of this algorithm, so-called "Communication-Avoiding *QR*" (CAQR) [43]. As in ScaLAPACK, the basic operation of CAQR is the factorization of a panel followed by the update of the trailing submatrix. Since the latter is dictated by the former, we focus on the panel factorization step called "Tall and Skinny *QR*" (TSQR). TSQR splits the initial $m \times b$ matrix into block-rows so-called *domains*. The domains are factorized independently from one another. A single reduction is thus required, inducing only $\log_2(P)$ messages during the whole panel factorization if there is a perfect matching between processors and domains. Fig. 8 illustrates the TSQR algorithm.

TSQR is particularly well adapted to the factorization of tall and skinny matrices, *i.e.*, matrices satisfying $M \gg N$ (see [43]). TSQR is an important kernel for two reasons. First, the *QR* factorization of tall and skinny matrices is directly used in several important applications of dense linear algebra such as block iterative methods, each time they need to compute an orthogonal basis of a set of vectors. Second, TSQR is the panel factorization of CAQR, which allows to process general matrices.

5.2.2. Implementation of TSQR with QCG-OMPI

We implemented the TSQR algorithm on top of ScaLAPACK. The local *QR* factorization occurring in a domain consists of a call to PDGEQRF ScaLAPACK's parallel routine. Such a call uses a subgroup of processes. We articulate TSQR with QCG-OMPI as follows. Because ScaLAPACK induces more messages than TSQR, we use the JobProfile to request a low latency among processes of a same subgroup and we accept a lower network connectivity between subgroups. This formulation actually corresponds to the classical clusters of clusters approach. To facilitate load balancing, we furthermore request a similar computing power between the groups. The meta-scheduler will allocate resources in the physical grid that match these requirements and expose them to the application through two-dimensional arrays of group identifiers. The application then creates one MPI communicator per group, using the *MPI_Comm_split* routine. TSQR can thus benefit from the topology of the grid: local factorizations are performed within clusters using ScaLAPACK whereas a single reduction occurs between clusters. The number of messages exchanged on the grid is thus limited. This is critical since inter-cluster latencies are two orders of magnitude greater than intra-cluster latencies (see Tables 1 and 2). And this ratio can even reach three or four orders of magnitudes on a grid built on top of Internet.

5.2.3. Experimental results

We conducted an experimental study using four clusters located in Bordeaux, Orsay, Sophia-Antipolis and Toulouse. In all the experiments reported in this study, we run one process per processor (thus two processes per node) using the serial GotoBLAS BLAS library.

Fig. 9(a) shows the performance of ScaLAPACK's *QR* factorization of matrices of width $N = 64$. Except for very tall matrices ($M > 10{,}000{,}000$), we observe a slow down of ScaLAPACK when using multiple sites. For very tall matrices, the amount of computation becomes so high that communications (which do not depend on the number of rows) become less critical; a slight speed up is eventually observed (right-most part of the graphs). However, even for the largest matrix considered in this study ($M = 33{,}554{,}432$, corresponding to 16 GB of data), a speed up of 2 is hardly reached while using four clusters.

The performance of TSQR articulated with QCG-OMPI depends on the number of domains used. The requirements asked through the JobProfile constraint to get at least one domain per cluster. But this is only a lower bound; it is possible to require more domains per cluster. In Fig. 9(b), we report the TSQR performance for the optimum number of domains per cluster. Contrary to ScaLAPACK, TSQR achieves a speed up on a significant range of matrix sizes. Indeed, for matrices of moderate to great height ($M \geq 500{,}000$), the fastest execution is the one conducted on all four sites. Furthermore, for very tall matrices (right-most part or the graphs), the speed up is optimum (almost 4 on all four clusters). In other words, QCG-TSQR enables a scalable and efficient *QR* factorization of large-scale tall and skinny matrices. Such factorization are repeatedly used in block iterative solvers for example. Finally, Fig. 9(c) shows that TSQR is significantly faster than ScaLAPACK.

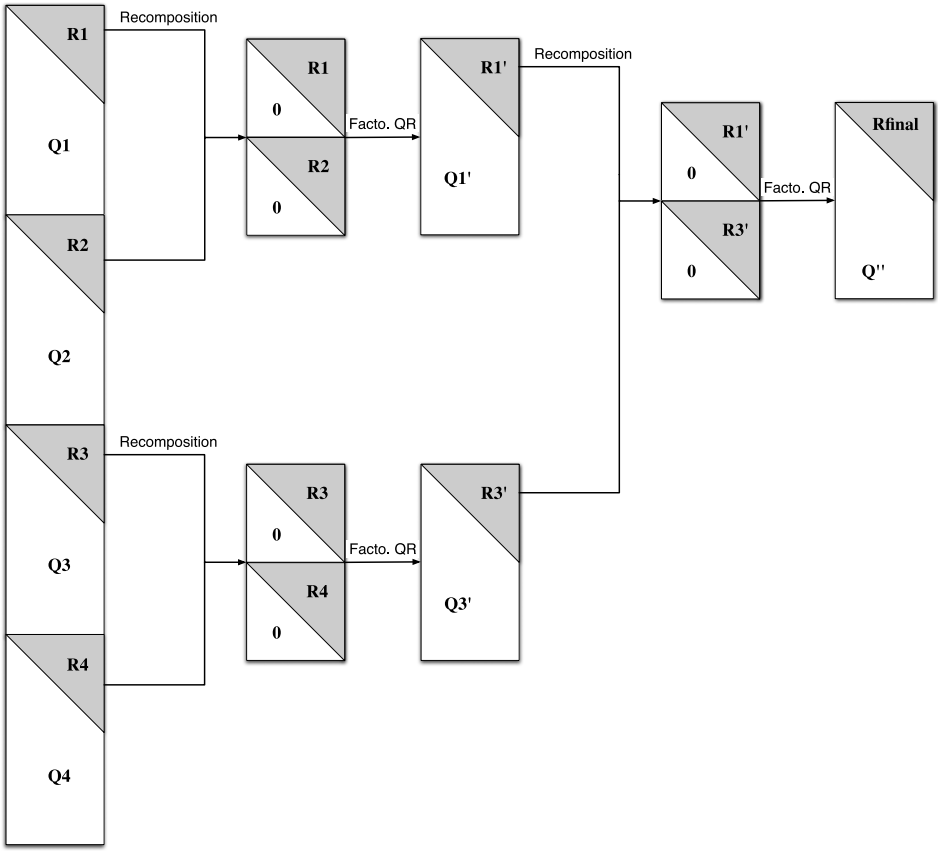
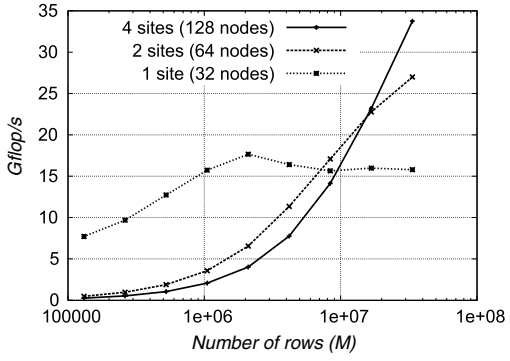
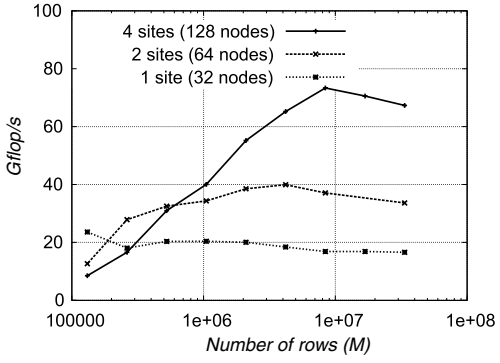


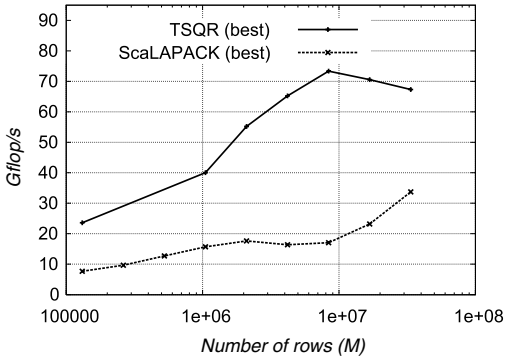
Fig. 8. TSQR algorithm. The initial tall and skinny matrix is split into four domains factorized independently from one another (left). The upper triangular matrices (R_i) are then assembled by pairs and factorized following a binary tree reduction (middle) until the final triangular factor R_{final} is obtained (right).



(a) ScaLAPACK.



(b) QCG-TSQR.



(c) QCG-TSQR vs ScaLAPACK. For each algorithm, the performance of the optimum configuration (one, two or four sites) is reported.

Fig. 9. Performance of QCG-TSQR and ScaLAPACK with a matrix of width $N = 64$.

6. Conclusion

In this paper, we have presented QCG-OMPI, an MPI communication and run-time environment designed specifically for grids. Based on Open MPI [12], it makes use of a grid-level extension of its run-time environment to address connectivity issues between administrative domains. Basic and advanced connectivity techniques are provided, and their performance has been evaluated and compared with other grid-enabled middleware.

Being able to communicate efficiently throughout the grid is not sufficient to ensure good performance of parallel applications, regarding the orders of magnitude between communication costs. As a matter of fact, applications must follow adapted, hierarchical communication patterns that match the physical topology of the resources they are executed on.

In the second part of this paper we have presented a set of collective applications adapted to the grid following a hierarchical approach in order to limit high latency communications. Moreover, the hierarchization of the patterns followed by these algorithms allows the usage of local, highly optimized algorithms.

In the last part of this paper we have explained a method to obtain good performance with parallel application on the grid making use of a contribution from the grid meta-scheduler. The communication patterns of the application are described as its virtual topology and submitted to the grid meta-scheduler. As a consequence, the complexity of the adaptation between allocated resources and the application is pushed to the scheduler. An extension of the MPI standard gives the possibility to retrieve this topology at run-time.

We have shown that this approach can be easily followed to design efficient applications with two typical, parallel applications. The first one is using a set of collective communications and a master–worker-based computation core. This pattern is widely used for parallel applications and we have shown that it can reach a significantly higher scalability following hierarchical patterns. The second application is a numerical linear algebra kernel that computes a matrix factorization. That latter study showed that QCG-OMPI enabled a scalable factorization of large-scale tall and skinny matrices.

Future works Directions for future works include improving the resiliency and availability of the grid infrastructure, for example using a highly distributed infrastructure (distributed broker) [44]. Besides, the positive results on parallel applications are strongly encouraging further research on grid-enabled applications.

Acknowledgements

The authors thank Emmanuel Jeannot for the interesting discussions about collective operations, and Stephane Genaud and Marc Grunberg for their help on Ray2mesh. Special thanks are due to George Bosilca for his explanations about the implementation of collective operations in Open MPI. The authors thank Laura Grigori for the stimulating discussions and her constructive remarks on communication-avoiding algorithms in linear algebra.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

Part of the first author's work was done in the context of a Research Assistant position in the Department of Electrical Engineering and Computer Science at the University of Tennessee.

The second author's work was partly supported by the EC grant for the QosCosGrid project (grant number: FP6-2005-IST-5 033883).

The fourth author's work was partly supported by NSF-CC (grant #811520).

References

[1] S.M. Larson, C.D. Snow, M. Shirts, V.S. Pande, Folding@home and genome@home: using distributed computing to tackle previously intractable problems in computational biology, 2009. arxiv.org/abs/arxiv:0901.0866.

[2] R. Bolze, Analyse et déploiement de solutions algorithmiques et logicielles pour des applications bioinformatiques à grande échelle sur la grille, Ph.D. Thesis, École Normale Supérieure de Lyon, October 2008.

[3] B. Allen, C. Christensen, N. Massey, T. Aina, M.A. Marquina, Network computing with einstein@home and climateprediction.net, Technical Report, CERN, Geneva, July 2005, CERN, Geneva, 11 Jul 2005.

[4] E. Walker, Benchmarking amazon EC2 for high-performance scientific computing, USENIX Login 33 (5) (2008) 18–23.

[5] C. Coti, A. Rezmerita, T. Herault, F. Cappello, Grid services for MPI, in: ACM/IEEE (Ed.), Proceedings of the 14th European PVM/MPI Users' Group Meeting, EuroPVM/MPI, Paris, France, October 2007, pp. 393–394.

[6] C. Coti, T. Herault, S. Peyronnet, A. Rezmerita, F. Cappello, Grid services for MPI, in: ACM/IEEE (Ed.), Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid, CCGrid'08, Lyon, France, May 2008, pp. 417–424.

[7] V. Kravtsov, D. Carmeli, W. Dubitzky, K. Kurowski, A. Schuster, Grid-enabling complex system applications with qoscosgrid: an architectural perspective, in: H.R. Arabnia (Ed.), Proceedings of the 2008 International Conference on Grid Computing & Applications, GCA 2008, Las Vegas, Nevada, USA, July 14–17, 2008, CSREA Press, 2008, pp. 168–174.

[8] P. Bar, C. Coti, D. Groen, T. Herault, V. Kravtsov, A. Schuster, M. Swain, Running parallel applications with topology-aware grid middleware, in: 5th IEEE International Conference on e-Science, eScience 2009, December 2009.

[9] E. Agullo, C. Coti, J. Dongarra, T. Herault, J. Langou, *QR* factorization of tall and skinny matrices in a grid computing environment, in: 24th IEEE International Parallel & Distributed Processing Symposium, IPDPS'10, Atlanta, Ga, April 2010.

[10] C. Coti, T. Herault, F. Cappello, MPI applications on grids: a topology-aware approach, in: LNCSm (Ed.), Proceedings of the 15th European Conference on Parallel and Distributed Computing, EuroPar'09, in: Delft, the Netherlands, August 2009.

[11] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, J. Dongarra, MPI: The Complete Reference, MIT Press, Cambridge, MA, USA, 1996.

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

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

[14] E. Gabriel, M.M. Resch, T. Beisel, R. Keller, Distributed computing in a heterogeneous computing environment, in: Vassil N. Alexandrov, Jack Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, 5th European PVM/MPI Users' Group Meeting, Liverpool, UK, September 7–9, 1998, Proceedings, in: LNCS, vol. 1497, Springer, 1998, pp. 180–187.

[15] Edgar Gabriel, Michael Resch, Thomas Beisel, Rainer Keller, Distributed Computing in a Heterogenous Computing Environment, 1998.

[16] I. Foster, N. Karonis, A grid-enabled MPI: message passing in heterogeneous distributed computing systems, in: High Performance Networking and Computing, SC98, IEEE/ACM, 1998.

[17] N.T. Karonis, B.R. Toonen, I.T. Foster, MPICH-G2: a grid-enabled implementation of the message passing interface, CoRR, cs.DC/0206040, 2002.

[18] I.T. Foster, Globus toolkit version 4: software for service-oriented systems, J. Comput. Sci. Technol 21 (4) (2006) 513–520.

[19] F. Gregoretti, G. Laccetti, A. Murli, G. Oliva, U. Scafuri, MGF: a grid-enabled MPI library, Future Gener. Comput. Syst. 24 (2) (2008) 158–165.

[20] W.L. George, J.G. Hagedorn, J.E. Devaney, IMPI: making MPI interoperable, April 25, 2000.

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

[22] R. Thakur, W. Gropp, Improving the performance of collective operations in MPICH, in: J. Dongarra, D. Laforenza, S. Orlando (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, 10th European PVM/MPI Users' Group Meeting, Venice, Italy, September 29–October 2, 2003, Proceedings, in: LNCS, vol. 2840, Springer, 2003, pp. 257–267.

[23] R. Rabenseifner, Optimization of collective reduction operations, in: Marian Bubak, G. Dick van Albada, Peter M.A. Sloot, Jack Dongarra (Eds.), Computational Science — ICCS 2004, 4th International Conference, Kraków, Poland, June 6–9, 2004, Proceedings, Part I, in: LNCS, vol. 3036, Springer, 2004, pp. 1–9.

[24] F. Cappello, P. Fraigniaud, B. Mans, A.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, San Francisco, CA, USA, April 2001, IEEE Computer Society, Los Alamitos, CA, 2001, p. 42.

[25] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, S. Vadhiyar, Numerical libraries and the grid: the GrADS experiments with ScaLAPACK, International Journal of High Performance Computing Applications 15 (4) (2001) 359–374.
[26] S.-S. Boutammine, D. Millot, C. Parrot, An adaptive scheduling method for grid computing, in: W.E. Nagel, W.V. Walter, W. Lehner (Eds.), 12th International Euro-Par Conference, Euro-Par'06, Dresden, Germany, August–September, in: LNCS, vol. 4128, Springer-Verlag, Berlin, New York, 2006, pp. 188–197.
[27] S. Genaud, A. Giersch, F. Vivien, Load-balancing scatter operations for grid computing, Parallel Comput. 30 (8) (2004) 923–946.
[28] C. Cérin, J.-C. Dubacq, J.-L. Roch, Methods for partitioning data to improve parallel execution time for sorting on heterogeneous clusters, in: Y.-C. Chung, J.E. Moreira (Eds.), Proceedings of the First International Conference on Grid and Pervasive Computing, GPC'06, May, in: LNCS, vol. 3947, Springer, 2006, pp. 175–186.
[29] R.A. Van Engelen, K.A. Gallivan, The gSOAP toolkit for web services and peer-to-peer computing networks, in: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'02, Washington, DC, USA, May 2002, IEEE Computer Society, 2002, pp. 128–135.
[30] A. Rezmerita, T. Morlier, V. Néri, F. Cappello, Private virtual cluster: infrastructure and protocol for instant grids, in: W.E. Nagel, W.V. Walter, W. Lehner (Eds.), Proceedings of the 12th European Conference on Parallel and Distributed Computing, EuroPar'06, in: LNCS, vol. 4128, Springer, 2006, pp. 393–404.
[31] C. Coti, T. Herault, F. Cappello, MPI applications on grid: a topology-aware approach, Technical Report 6633, INRIA, September 2008.
[32] S. Lacour, C. Pérez, T. Priol, A network topology description model for grid application deployment, in: GRID, 2004, pp. 61–68.
[33] Valentin Kravtsov, Pavel Bar, David Carmeli, Assaf Schuster, Martin Swain, A scheduling framework for large-scale, parallel, and topology-aware applications, Journal of Parallel and Distributed Computing (2010) 983–992.
[34] V. Kravtsov, D. Carmeli, A. Schuster, B. Yoshpa, M. Silberstein, W. Dubitzky, Quasi-opportunistic supercomputing in grids, hot topic paper, in: IEEE International Symposium on High Performance Distributed Computing, Monterey Bay California, USA, 2007.
[35] V. Kravtsov, D. Carmeli, W. Dubitzky, A. Orda, A. Schuster, M. Silberstein, B. Yoshpa, Quasi-opportunistic supercomputing in grid environments, in: A.G. Bourgeois, S.-Q. Zheng (Eds.), Proceedings of the 8th International Conference in Algorithms and Architectures for Parallel Processing, ICA3PP 2008, in: LNCS, vol. 5022, Springer, 2008, pp. 233–244.
[36] V. Kravtsov, M. Swain, U. Dubin, W. Dubitzky, A. Schuster, A fast and efficient algorithm for topology-aware coallocation, in: M. Bubak, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Proceedings of the 8th International Conference on Computational Science, ICCS 2008, Part I, in: LNCS, vol. 5101, Springer, 2008, pp. 274–283.
[37] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Vicat-Blanc Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, B. Quetier, O. Richard, Grid'5000: a large scale and highly reconfigurable grid experimental testbed, in: SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing CD, Seattle, Washington, USA, November 2005, IEEE/ACM, 2005, pp. 99–106.
[38] P.K. McKinley, Y.-J. Tsai, D.F. Robinson, Collective communication in wormhole-routed massively parallel computers, IEEE Computer 28 (12) (1995) 39–50.
[39] M. Grunberg, S. Genaud, C. Mongenet, Parallel seismic ray tracing in a global earth model, in: Hamid R. Arabnia (Ed.), Las Vegas, Nevada, USA, June, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'02, vol. 3, CSREA Press, 2002, pp. 1151–1157.
[40] G.H. Golub, C.F. Van Loan, Matrix Computations, 3 edition, Johns Hopkins University Press, Baltimore, USA, 1996.
[41] R. Schreiber, C. Van Loan, A storage efficient *WY* representation for products of Householder transformations, SIAM J. Sci. Stat. Comput. 10 (1) (1989) 53–57.
[42] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley, ScaLAPACK Users' Guide, SIAM, 1997.
[43] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-avoiding parallel and sequential *QR* factorizations, CoRR, abs/0806.2159, 2008.
[44] Roberto Podestá, Victor Iniesta, Ala Rezmerita, Franck Cappello, An information brokering service provider (ibsp) for virtual clusters, in: Proceedings of the On the Move to Meaningful Internet Systems (OTM) Conference, in: Lecture Notes in Computer Science, vol. 5870, Springer, 2009, pp. 165–182.

Emmanuel Agullo received a Ph.D. from École Normale Supérieure de Lyon, worked at University of Tennessee as a postdoctoral research associate and recently joined INRIA as a junior research scientist. His core expertise lies in high performance computing, more specifically numerical linear algebra algorithms for sparse and dense direct solvers.

Camille Coti prepared a Ph.D. at INRIA Saclay-Île de France while being a frequent visitor at the University of Tennessee, and graduated in Nov. 2009 from the University of Paris South-XI. She is now an assistant professor at the University of Paris North-XIII. She works on parallel computing and tools for parallel computing.

Thomas Herault is associate professor at the Paris South University, currently delegated at INRIA, and visiting researcher at the University of Tennessee. He defended his Ph.D. on the mending of transient failure in self-stabilizing systems under the supervision of Joffroy Beauquier. He is a member of the Grand-Large INRIA team, in which he led the MPICH-V project. He currently works on fault-tolerant protocols in large-scale distributed systems, programming paradigms in High Performance Computing, and Grids.

Julien Langou received two M.Sc. Degrees from the Ecole National Superieure de l'Aeronautique et de l'Espace (SUPAERO), Toulouse, France in 1999: one in propulsion engineering and one in applied mathematics. In 2003, he received his Ph.D. in applied mathematics from the National Institute of Applied Sciences (INSA), Toulouse, France, for his work at the CERFACS laboratory (Toulouse). He then worked at the Innovative Computing Laboratory at the University of Tennessee until 2006, becoming a Senior Scientist. He now holds an appointment as assistant professor in the Department of Mathematical and Statistical Sciences at the University of Colorado Denver. His research interest is in numerical linear algebra with application in high performance computing.

Sylvain Peyronnet was born in 1977. He is currently Maitre de Conferences at LRI, Université Paris-Sud XI. His research interests are approximate verification, probabilistic algorithms, numerical methods and robust and fault-tolerant computing.

Ala Rezmerita defended her Ph.D. on September 2009 under the supervision of Franck Cappello at the University of Paris Sud (France). Ala Rezmerita worked on the grid middleware and Desktop Grid in the Laboratoire de Recherche en Informatique (LRI) in the Parallelism Team. Now, Ala Rezmerita is holding a postdoc position in the Parallelism research group of University of Paris Sud, where she is working on a secured OS project aiming ordinary users.

Franck Cappello before establishing the INRIA-Illinois joint laboratory on PetaScale Computing, established and led the INRIA Grand-Large project during 6 years, focusing on high performance issues in large-scale distributed computing systems. He also initiated and was the director of the Grid5000 project until 2009, a nationwide computer science platform for research in large-scale distributed systems. He leaded the XtremWeb (desktop grid) and MPICH-V (fault-tolerant MPI) projects and authored more than 100 papers in the domains of high performance programming, desktop grids, and fault tolerant MPI. He is now co-director of the INRIA-Illinois joint laboratory on PetaScale Computing and adjunct scientific director of ALADDIN/Grid'5000, the new four-year INRIA project aiming to sustain the Grid5000 infrastructure and to open it to cloud computing, service infrastructure, and the future internet research. He is one of the leaders of the Exascale effort through his participation to IESP and several other Exascale initiatives. He is an editorial board member of the international Journal on Grid Computing, Journal of Grid and Utility Computing, and Journal of Cluster Computing. He is a steering committee member of IEEE HPDC and IEEE/ACM CCGRID.

Jack Dongarra holds an appointment as University Distinguished Professor of Computer Science in the Electrical Engineering and Computer Science Department at the University of Tennessee and holds the title of Distinguished Research Staff in the Computer Science and Mathematics Division at Oak Ridge National Laboratory (ORNL), Turing Fellow in the Computer Science and Mathematics Schools at the University of Manchester, and an Adjunct Professor in the Computer Science Department at Rice University. He specializes in numerical algorithms in linear algebra, parallel computing, use of advanced-computer architectures, programming methodology, and tools for parallel computers. His research includes the development, testing and documentation of high quality mathematical software. He has contributed to the design and implementation of the following open-source software packages and systems: EISPACK, LINPACK, the BLAS, LAPACK, ScaLAPACK, Netlib, PVM, MPI, NetSolve, Top500, ATLAS, Open-MPI, and PAPI. He has published approximately 300 articles, papers, reports and technical memoranda and he is coauthor of several books. He was awarded the IEEE Sid Fernbach Award in 2004 for his contributions in the application of high performance computers using innovative approaches and in 2008 he was the recipient of the first IEEE Medal of Excellence in Scalable Computing. He is a Fellow of the AAAS, ACM, IEEE, and SIAM and a member of the National Academy of Engineering.