Comparison and tuning of MPI implementations in a grid context

Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, Stéphane Genaud, Pascale Vicat-Blanc Primet

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n^o 5668

Comparison and tuning of MPI implementations in a grid context

Ludovic Hablot,
Olivier Glück,
Jean-Christophe Mignot,
Stéphane Genaud,
Pascale Vicat-Blanc Primet

May 2007

Research Report N^o RR2007-21

Comparison and tuning of MPI implementations in a grid context

Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, Stéphane Genaud, Pascale Vicat-Blanc Primet

May 2007

Abstract

Today, clusters are often interconnected by long distance networks within grids to offer a huge number of available ressources to a range of users. MPI, the standard communication library used to write parallel applications, has been implemented for clusters. Two main features of grids: long distance networks and technological heterogeneity, raise the question of MPI efficiency in grids.
This report presents an evaluation of four recent MPI implementations (MPICH2, MPICH-Madeleine, OpenMPI and GridMPI) in the french research grid: Grid'5000. The comparison is based on the execution of pingpong, NAS Parallel Benchmarks and a real application in geophysics. We show that this implementations present performance differences. Executing MPI applications on the grid can be beneficial if the parameters are well tuned. The paper details the tuning required on each implementation to get the best performances.

Keywords: MPI, MPI implementations, Grid'5000, TCP tuning, MPI optimizations

Résumé

Aujourd'hui les clusters sont souvent interconnectés par des réseaux longues distance à l'intérieur des grilles, dans le but d'offrir une plus grande capacité de calcul. MPI, la bibliothèque standard utilisée pour écrire des applications parallèles, a été concue pour les clusters. Cependant la grille apporte deux nouvelles contraintes : les réseaux longue distance et l'hétérogénéité. Celles-ci posent la question de l'efficacité de MPI dans ce contexte.
Ce rapport présente une évaluation de quatre implémentations MPI récentes (MPICH2, MPICH-Madeleine, OpenMPI and GridMPI) dans la grille de recherche francaise Grid'5000. La comparaison est basée sur l'execution d'un pingpong, des NAS Parallel Benchmarks et d'une application réelle de géophysique. Nous montrons que les implémentations présentent des différences de performance. L'exécution d'applications MPI sur la grille peut être bénéfique si l'on effectue un paramètrage correct. Ce rapport présente également les optimisations nécessaires sur chaque implémentation MPI pour obtenir de meilleures performances.

Mots-clés: MPI, implementations MPI, Grid'5000, paramètrage de TCP, optimisations de MPI

1 Introduction

Today, clusters are often interconnected by long distance networks within grids to offer a huge number of available ressources to a range of users. MPI is the standard communication library used to write parallel applications.

Users want to execute their applications, written for clusters, on grids to get better performance by using more resources. For example, geophysical applications like ray2mesh [14], a seismic ray tracing in a 3D mesh of the Earth, or medical applications like Simri [5], a 3D Magnetic Resonance Imaging (MRI) simulator, have been executed on grids with success. However, MPI implementations are initially written for clusters and do not consider the specificities of grid interconnections.

The grid raises mainly three problems to execute MPI applications. First, MPI implementations have to manage efficiently the long-distance between sites. Actually, the high latency between sites is very costly, especially for little messages. Inter-sites communications take more time than intra-sites ones. In a grid, inter-sites links may offer higher capacities than cluster links.

Secondly, MPI implementations have to manage heterogeneity of the different high speed networks composing the grids, i.e. enable intra-site communications between them (for example, a Myrinet interconnect and an Infiniband cluster on the same site) and also inter-sites communications between them and the WAN (Wide Area Network). We do not address this problem here. In our experiments, all the communications use TCP.

Finally, the CPUs heterogeneity has an impact on the overall application performance execution. Indeed, it could be of interest to take into account this heterogeneity in the task placement phase.

Several MPI implementations are taking some characteristics of grids into account, to offer better execution of MPI applications on grids. The MPICH2 [16], GridMPI [15], MPICH-Madeleine [1], OpenMPI [13], MPICH-G2 [24] implementations can be used on a grid but are not similarly optimized to this specific context. The goal of this paper is to identify which ones are efficient on a real grid and in which conditions. An other question is to define how to best configure these implementations to achieve good performance on the grid. Finally, it could be usefull to have a better view of which communication patterns better fit grid context. To answer these questions, we conduct experiments in Grid'5000 [6] a french research grid platform gathering more than 3000 processors. Nine sites are connected by a dedicated WAN at 1 or 10 Gbps. This architecture provides researchers a fully reconfigurability feature to dynamically deploy and configure any OS on any host.

This paper is organised as follows. The Section 2 presents a state of the art of MPI implementations usable in a grid context. In the Section 3, we describe the experimental protocol. The section 4 presents the results of our experiments and the necessary optimizations required to achieve good performances. Finally, we conclude and present future work.

2 MPI implementations and applications for the grid

In this section, we first present some MPI implementations which address one or more specificities of grids. Then, we give examples of related works executing MPI applications on a grid.

2.1 MPI implementations for cluster of clusters and grids

This section presents the following implementations that improve either network heterogeneity or long-distance communications: GridMPI, OpenMPI, MPICH-Madeleine, MPICH-G2, MPICH-VMI. We also describe MPICH2 because it is a largely used implementation. Moreover, a lot of MPI implementations are based on the MPICH implementation.

2.1.1 MPICH2[16]

This implementation is the successor of MPICH[17], used by a lot of people. It is not a grid implementation but could be used in a grid if all the communications use TCP. It has a four layered architecture which could be quite easily modified. Results with this implementation are well-known that is why we use it as a reference in our tests.

2.1.2 MPICH-Madeleine[1]

This implementation is based both on MPICH[17] and Madeleine[2], a communication library which allows communications between several commonly used high speed networks. Thus, MPICH-Madeleine supports efficiently networks heterogeneity between TCP, SCI, VIA, Myrinet MX/GM and Quadrics. MPICH-Madeleine uses a gateway between two clusters interconnected by different networks. For example, if you have a cluster A using Myrinet, a cluster B using SCI and a node shared by the two clusters, messages can be sent between them directly through Madeleine and without using TCP. It seems that MPICH-Madeleine is the implementation that manages network heterogeneity the most efficiently because it has been designed especially for cluster of clusters. It can be used in a grid context using TCP for long-distance communications.

2.1.3 OpenMPI[13]

OpenMPI is an open source MPI-2 compliant implementation built from scratch combining strong points and resources from earlier projects (namely FT-MPI, LA-MPI, LAM/MPI and PACX-MPI) as well as new concepts. Open MPI can be considered as the combination of ORTE and the MPI library itself.

ORTE, the Open Run Time Environment, is a spin-off from the original OMPI project. It's main objective is to provide support for the execution of large scale heterogenous applications. It is an important part of the transparency offered by OMPI.

The OpenMPI library is a component based architecture that allows people to easily add their modules in Open MPI. Progression of communications is handled by the PML (Point-to-point Management Layer) module. This module can use different BTL (Byte Transfer Layer), one for each protocol, hence providing network heterogeneity. Supported protocols are TCP, Myrinet MX/GM, Infiniband OpenIB/mVAPI. Long distance networks characteristics are not particularly taken into account.

The OpenMPI library automatically handles grid specific configurations. For example, clusters portals and cluster specific network fabrics are automatically detected. Nevertheless, specific tunings are still required to achieve optimal performance of long and fast networks of clusters in most cases.

2.1.4 GridMPI[15]

This implementation is specifically designed to be executed in a grid environnement. The support of heterogeneity is done with IMPI (Interoperable MPI). IMPI is an overlay over other MPI implementations. Each cluster could use a dedicated VendorMPI to exploit available high speed interconnects and GridMPI is doing the link between them using IMPI. GridMPI has been designed to optimize long-distance communications in MPI applications. Modifications are done at two levels, in the TCP layer and in MPI collective communications level. First, at start time, GridMPI designers propose to modify TCP as follow:

- adding pacing mechanism to avoid burst effects[30]
- reducing RTO (Retransmit Timeout)[20]
- using two different congestion window sizes for collective and point-to-point communications[20]
- bypassing the socket layer [22]

For now, the pacing mechanism only is available.

Secondly, GridMPI integrates modifications of algorithms for collective operations (see [21]): in a grid context, communications between clusters have a higher bandwidth than intra-cluster once. So, the adapted algorithms of GridMPI (based on Van de Geijn (Bcast) and Rabenseifner (Allreduce) algorithms) use multiple node-to-node connections to send data on the long-distance links. Thus, the MPI_Bcast and the MPI_Allreduce primitives of GridMPI offer better performances on the grid.

2.1.5 MPICH-G2[24]

MPICH-G2 is based both on MPICH and Globus Toolkit[12]. MPICH is described above, Globus provides services to manage security (including a certificate management), execution and data in the grid. Thanks to Globus, MPICH-G2 allows to allocate and manage grid ressources. MPICH-G2 creates only one sockets from one process to another one and this socket is used in a bidirectionnal mode. It includes a support for large messages using several TCP streams. This mechanism is supported by GridFTP and a few code has to be added in application. Only TCP is available for intersites communications. The heterogeneity of high-speed networks is managed by VendorMPI.

The collective operations are "topology-aware" i.e. they take into account the network architecture to communicate: WAN $<$ LAN $<$ intra-machine TCP $<$ Vendor MPI (from the slowest to the fastest). There is a mechanism to discover the topology of the network. For now, only MPI_Gatherv, MPI_Scatterv and MPI_Alltoallv are not "topology-aware".

2.1.6 MPICH-VMI

MPICH-VMI is based both on MPICH and VMI [25]. VMI is a middleware communication layer that addresses the issues of availability, usability, and management in a grid context. It supports different high speed networks: TCP/IP, Myrinet GM, Infiniband VAPI/OpenIB/IBAL. The communications patterns of an application are stored in a database in order to use it to realise a task placement. This feature is not implemented yet. Collective operations are optimized to avoid long-distance communications.

2.1.7 Synthesis

Table 1: Comparison of MPI implementation features

	Long-distance optimizations	**Network heterogeneity management**	**First / Last publication**
MPICH2	None	None	2002 [16] / 2006 [8]
GridMPI	Optim. of TCP Optim. of Bcast and All_reduce	Heterogeneity using IMPI above TCP but no support of low latency high-speed networks	2004 [22] / 2006 [21]
MPICH-Madeleine	None	Gateway between high-speed networks Support of TCP, SCI, VIA, Myrinet MX/GM, Quadrics	2003 [1] / 2007 [3]
OpenMPI	None	Gateway between high-speed networks Support of TCP, Myrinet MX/GM, Infiniband OpenIB/mVAPI	2004 [13] / 2007 [9]
MPICH-G2	Optim. of collective operations Using parallel streams for big messages	Using TCP above VendorMPI	2003 [24]
MPICH-VMI	Optim. of collective operations	Gateway between high-speed networks Support of TCP/IP, Myrinet GM Infiniband VAPI/OpenIB/IBAL	2002 [25] / 2004 [26]

Table 1 summarizes the main features of each MPI implementation. Three of these implementations can manage network heterogeneity efficiently using gateways (MPICH=Madeleine, OpenMPI and MPICH-VMI). For GridMPI and MPICH-G2, the management of the heterogeneity is done using IMPI and an appropriate VendorMPI. This may introduce a large overhead in latency. For long-distance communications, the optimizations are done at two levels. First, MPICH-G2, MPICH-VMI or GridMPI optimize some or every collective operations to be efficient on the grid. Secondly, GridMPI modifies the TCP behavior to adapt it to the high long-distance latency.

MPICH-VMI and MPICH-Madeleine are similar excepted that MPICH-Madeleine does not optimize collective operations. However, we prefer to use MPICH-Madeleine only, because this implementation is still "in-live" compared to the MPICH-VMI. MPICH-G2 has a heavy certificate management due to Globus, quite hard to install, but the optimizations for long-distance communications seem to be interesting.

We choose MPICH2 as a reference, and GridMPI for its optimizations on long distance communications. Finally, our study is based on these four implementations: MPICH2, GridMPI, MPICH-Madeleine and OpenMPI.

Some works have been done to compare features of MPI implementations. In [23], the authors compare different methods used in MPI implementations (PACX-MPI, MPICH-G2, Stampi and MagPIe) to communicate between a private and a public network. Performance measurement aims at evaluating these different approaches. MPI experiments are done with PACX-MPI. They do not compare the performances of each MPI implementations between them.

FT-MPI[10] or MPICH-V[7] are other MPI implementations for grids but they deal with fault-tolerance issues rather than the performance execution on the grid.

2.2 Example of MPI applications execution on the grid

In this section, we give two examples of application that have been evaluated on the grid.

2.2.1 Ray2mesh

This physics application is a software suite for seismology. ray2mesh [18] contains a seismic ray-tracer, a mesh library dedicated to geophysics and a mesh refinement tool. The goal of this suite is to build a seismic tomography model for the Earth. A seimic wave is modeled by a set of rays, where each ray represents the wave front propagation from the hypocenter to a recording station. The application consist in tracing the path of these rays in a regular mesh of the Earth. A ray data set may contain millions of rays. Therefore parallelizing the computation is a key issue. The parallel implementation of ray2mesh uses MPI.

In [14], the authors describe the experiments they have executed on Grid'5000. Their application have been deployed on a various number and distribution of nodes and executed using LAM/MPI. The speed up obtained when varying the number of nodes is linear, but the speed up of communication phase is constant if the number of nodes exceed 192. Indeed, this phase sends results to all nodes and this is very costly if there are a lot of nodes.

2.2.2 Simri

Simri[5] is an implementation of a Magnetic Resonance Imaging (MRI) simulator. From a 3D virtual object and a MRI sequence, the simulator computes a set of Radio Frequency that can be transform in a Magnetic Resonance image. During the computation phase, the master node is divides the work and send a set of vectors to each slave. The slaves compute the magnetization evolution of the vector and send the result to the master.

The experiments have been done on a cluster of 8 Pentium III, 1 Ghz. Its use MPICH-G2 and Globus 2.2. The synchronization and communication take only 1.5% of the total time if the size of the input object is greater than 256*256. The execution on a 8 nodes cluster gives an efficiency near of 100% i.e. the computation phase take seven times less time than the execution on one node (The master node does not compute).

3 Experimental protocol

3.1 Goals and Methodology

There are two main goals of our experiments. First, we want to execute different MPI implementations in a real grid context and to see their efficiency on the Grid'5000 grid platform. Then, we want to study the advantages of executing real MPI applications on the grid instead of a cluster.

To compare the different implementations, we run between two sites of the grid a simple MPI pingpong with the default configuration of each implementation. We compare the results obtained to those obtained between two nodes on the same cluster. We have also run a TCP pingpong to evaluate the MPI overhead of each implementation.

Table 2: NPB features

	Type of comm.	**Size and number of messages**
EP	P. to P.	192 * 8 B + 68 * 80 B
CG	P. to P.	126479 * 8 B + 86944 * 147 kB
MG	P. to P.	50809 * various sizes from 4 B to 130 kB
LU	P. to P.	1200000 * 960 B<msg<1040 B
SP	P. to P.	57744 * 45 kB<msg<54 kB + 96336 * 100 kB<msg<160 kB
BT	P. to P.	28944 * 26 kB + 48336 * 146 kB<msg<156 kB
IS	Collective	176 * 1 kB + 176 * 30 MB
FT	Collective	320 * 1 B + 352 * 128 kB

Since the MPI pingpong can not represent the behavior of real applications, we also run experiments with the NPB (NAS Parallel Benchmarks [4]) and one real MPI application (ray2mesh [18]).

Pingpong is our own program. One processus sends MPI messages with the MPI_Send primitive to another one which does a MPI_Recv and sends the message back. The size of the messages range from one byte to 64 MBytes. For each size, we obtain the round-trip latency. Then, we can plot the MPI bandwidth available between the two nodes.

The NPB are a set of eight programs (BT, CG, EP, FT, IS, LU, MG and SP) that gives a good panel of the different parallel applications that could be executed on a cluster or a grid. The NPB have been designed to compare performances of MPI implementations and clusters. The NPB use several classes (S, W, A, B, C, D) to represent the size of the problem. We have done our experiments with the B class on four or sixteen nodes. They have different kind of communication schemes. We can see in [29] that all of them are symmetric excepted CG and EP. In [11], the authors give the type of communications (point-to-point or collective) and the number of messages for each NAS for the class A on 16 nodes. We have run each NAS with a modified MPI implementation to find their communication pattern. Table 2 summarizes the communication features of NAS parallel benchmarks. EP mostly computes and sends very few data. CG communicates with little and big messages. MG sends varing size of messages. LU sends medium size messages and is the most communicating. BT and SP send a lot of big messages. Finally, IS and FT communicate using collective operations with very big messages for IS. FT uses the primitive MPI_Bcast and IS uses MPI_Allreduce, MPI_Alltoall and MPI_Alltoallv.

3.2 Testbed description

Our experiments are conducted on Grid'5000. Figure 1 shows the Grid'5000 backbone connecting the nine sites of this grid (Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia and Toulouse). For example, the RTT TCP latency is about 19 ms for the link Rennes-Sophia or 18.2 ms for Toulouse-Lille. Each site is composed of an heterogeneous set of nodes and local networks (Infiniband, Myrinet, Ethernet 1 or 10 Gbs). Sites are connected by a dedicated WAN operated by RENATER [27] at 1 or 10 Gbps. This architecture provides researchers with a fully reconfigurability feature to dynamically deploy and configure any OS on any host. This feature allows them to have administrator rights to change TCP parameters for instance.

Figure 2 shows the description of our testbed. We use between 1, 2 or 8 nodes on each cluster (Nancy and Rennes) depending on the experiment (execution of pingpong or NPB). Each node is connected to a switch with a 1 Gbps Ethernet card. Thus, the maximum bandwidth available between one process in Rennes and one process in Nancy is 1 Gbps. The RTT TCP latency is about 11.6 ms.

Table 3 shows the hardware and software specifications of the hosts.

We have used the following versions of MPI implementations: MPICH2 is the 1.0.5 release using default parameters. GridMPI is 1.1. We do not use the IMPI support as all the communications (intra and inter sites) are achieved by TCP. MPICH-Madeleine is the svn version of the 6th of December 2006. It uses thread (`-lib=-lpthread`) and fast buffering `-with-device= ch_mad:-fast-buffer`.

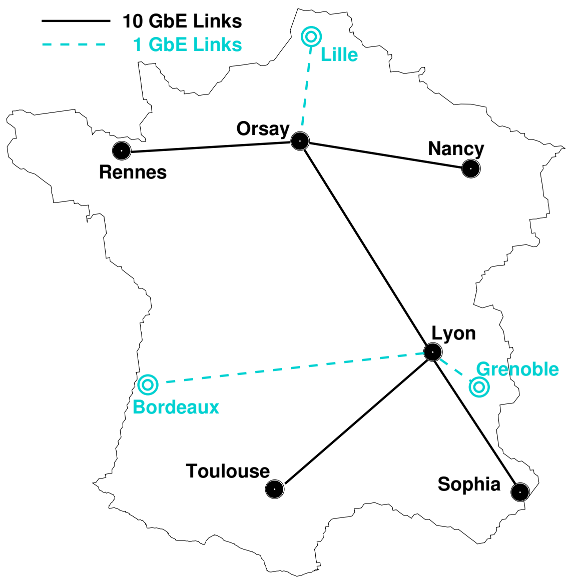


Figure 1: Grid'5000 backbone

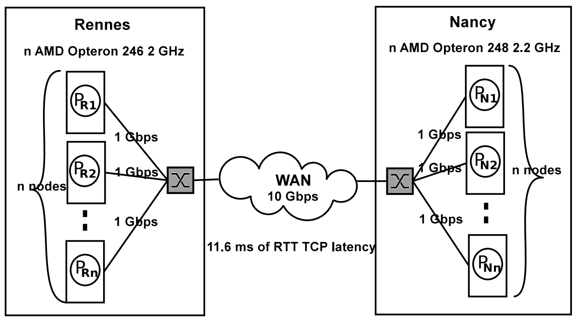


Figure 2: Configuration used

Table 3: Host Specifications

	Rennes	**Nancy**
Processor	AMD Opteron 248 2.2 GHz	AMD Opteron 246 2GHz
Motherboard	Sun Fire V20z	HP ProLiant DL145G2
Memory	2 GB	
NIC	1Gbps Eth	
OS	Debian	
Kernel	2.6.18	
TCP version	BIC + Sack	

Finally, OpenMPI version is 1.1.4.

4 Results of experiences and optimizations

4.1 First ping-pong results

We run our pingpong between two nodes. The results within a cluster are done between two nodes of the Rennes cluster corresponding to P_{R1} and P_{R2} on Figure 2. The experiments in the grid are done between P_{R1} and P_{N1}. Pingpong results are done with 200 round-trips with the default configuration of MPI implementations. Then, we take the minimum result (among the 200) for the latency and the maximum for the bandwidth in order to eliminate perturbations due to other Grid'5000 users.

Table 4 shows the latency comparison between the different MPI implementations in the Rennes cluster or in the grid (between Rennes and Nancy). The overhead of MPI implementations is almost the same in a cluster or in a grid excepted for MPICH-Madeleine. MPICH-Madeleine has a lower overhead in the grid. These figures confirm the long-distance overhead which corresponds to the network latency (11600 μs of RTT ping corresponding to 5800 μs one way).

Table 4: Comparison of latency in a cluster and in a grid (in us)

	In the Rennes cluster	**In the grid: btwn Rennes and Nancy**
TCP	41	5812
MPICH2	46 (+5)	5818 (+6)
GridMPI	46 (+5)	5819 (+7)
MPICH-Madeleine	62 (+21)	5826 (+14)
OpenMPI	46 (+5)	5820 (+8)

Figure 5 shows the bandwidth obtained on a cluster with the default parameters. All the implementations reach 940 Mbps that is the maximum goodput of TCP on a 1 Gbps link. Each implementation has a threshold around 128 kB excepted GridMPI. This threshold is due to the different modes of MPI as explained in section 4.2.2. Figure 3 represents the results of the same experiment on the grid. Results are very bad. None of the implementations nor direct TCP implementation of the pingpong reached a higher bandwidth than 120 Mbps. These behaviors are due to both bad TCP and MPI configuration. We describe in the next section how to tune them.

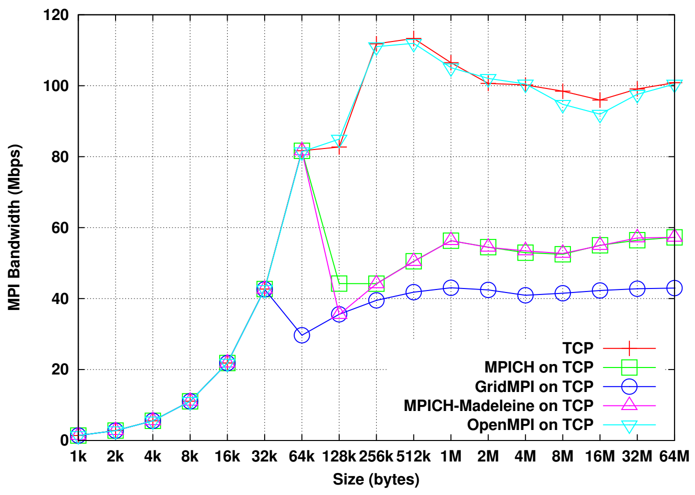


Figure 3: Comparison of MPI bandwidth obtained with different MPI implementations on a distant network (grid) with the default parameters

4.2 Tuning to be done on the grid

4.2.1 TCP Level

In TCP, communications are done through sockets. TCP uses a congestion window to limit the number of 'on the fly' packets that are sent but not acknowledged. The sender does not know that a packet is received until being acknowledged, so it must keep this packet in a buffer to restransmit it if needed. Hence, the

TCP socket buffers must be able to contain at least a congestion window. The size of these buffers has to be set at least at the product of RTT * bandwidth [28].

Two linux kernel variables allow to modify the size of these buffers.

- The first value specifies the maximum size that a socket can allocate to its buffer: */proc/sys/net/core/rmem_max* and *wmem_max* have to be increased (see [19]).
- The linux kernel can adapt these values dynamically with a mechanism called auto-tuning. The first and last value of */proc/sys/net/ipv4/tcp_rmem* and */proc/sys/net/ipv4/tcp_wmem* represent the bounds of auto-tuning in the linux kernel. The middle value is the size given initially by the kernel when a socket is created. The last value has to be increased to RTT * bandwidth.

With the first method, we have to modify the code of each implementation to set the initial size of the socket. With the second, the size is set automatically.

In our experiments, between Rennes and Nancy, the socket buffer has to be set to at least at 1.45 MB (RTT=11.6 ms, bandwidth=1 Gbps). However, for compatibility with the rest of the grid, we set it at 4 MB. This tunning is self sufficient for MPICH2 and MPICH-Madeleine but not for GridMPI and OpenMPI. In GridMPI, the middle value of TCP socket buffer has to be increased. In OpenMPI, the size of the buffers is specified when a socket is created. This size is 128 kB by default. The modification of this size is done dynamically, passing two parameters to mpirun: `-mca btl_tcp_sndbuf 4194304 -mca btl_tcp_rcvbuf 4194304`

Figure 6 shows the new pingpong result after TCP tuning and parameters modification in OpenMPI. The maximum MPI bandwidth available is now around 900 Mbps in the grid which is a good result regarding the 940 Mbps in the Rennes cluster. However, the half bandwidth is only reached around 1 MB in the grid against 8 kB in the cluster. Finally, GridMPI performances are better than the other MPI implementations and very near TCP ones. We still observe the threshold around 128 kB excepted for GridMPI. We will see in the next section why and how to tune MPI implementations for grid experiments in order to delete this threshold.

4.2.2 MPI implementations level

Figure 4, shows MPI communications paradigm. Two modes are used to communicate in MPI: the eager mode and the rendez-vous mode.

- the eager mode: the application data are copied in the TCP buffer. If the application message is very large, MPI fragments the message (Frag. 1 and Frag. 2 in Fig. 4) into several MPI data messages. Then, TCP sends the MPI fragments to the receiver and stores them in the reception socket buffer. If the matching MPI_Recv has been posted, the fragments are directly copied in the application buffer (arrow 1 in Fig. 4). Otherwise, an additionnal copy is done (arrow 2) in a tempory MPI buffer waiting for the MPI_Recv.
- the rendez-vous mode: a MPI control message (MPI_Request) informs the receiver that a MPI_Send in the rendez-vous mode has started. When the matching MPI_Recv is posted on the receiver, it sends back an MPI acknowledge for each fragment and data are directly copied in the application user buffer.

The threshold between the eager mode and the rendez-vous mode has to be set correctly. For small messages, the eager mode is preferred (the rendez-vous mode is more costly than the extra copy) whereas it is the opposite for large messages. Since the rendez-vous cost depends on the RTT, it is higher in the grid than in the cluster. Thus, it is necessary for grid experiments to adjust correctly the eager/rendez-vous threshold parameter in each MPI implementation.

The ideal threshold are shown in Table 5. The threshold provided are the optimal ones for messages under 64 MB. In the case of sending bigger messages, they shall be raised up. These thresholds consider that the application post the recv before the send, there is no copy out on the receiver side.

The eager/rndv threshold can be modified in this way:

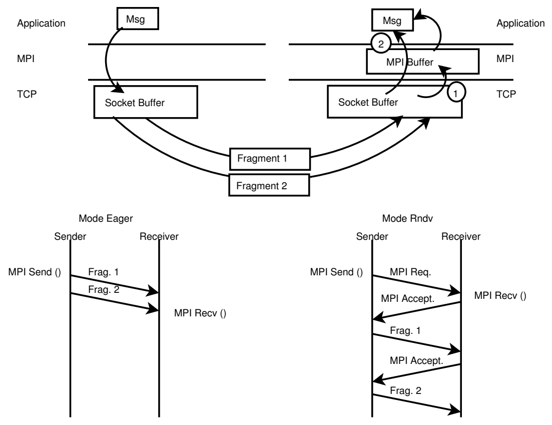


Figure 4: Interactions between TCP and MPI layers for the two MPI communication modes

Table 5: Ideal eager/rndv threshold for each implementation in a cluster or in the grid

	Original threshold	**Threshold in a cluster**	**Threshold in the grid**
MPICH2	256 kB	65 MB	65 MB
GridMPI	∞	-	-
MPICH-Madeleine	128 kB	65 MB	65 MB
OpenMPI	64 kB	32 MB	32 MB

- For MPICH2: modification of the file `./src/mpid/ch3/channels/sock /include/mpidi_ch3_post.h`: `#define MPIDI_CH3_EAGER_MAX_MSG_SIZE (65 * 1024 * 1024)`
- For GridMPI: modification of the environnement variable _YAMPI_RSIZE but by default GridMPI does not use the rendez-vous mode for MPI_Send: `export _YAMPI_RSIZE=1048576`
- For MPICH-Madeleine: modification of the file `./mpid/ch_mad/hot_stuff.h`: `#define DEFAULT_SWITCH (65 * 1024 * 1024)`
- For OpenMPI: modification is done with mpirun arguments: `-mca btl_tcp_eager_limit 33554432`

Figure 7 shows the pingpong results when we optimized both TCP and MPI implementations. All MPI implementations have almost the same performances excepted OpenMPI which has a lower bandwidth for big messages. TCP and these implementations have the same behavior. Thus, the TCP and MPI parameters we have set are optimal.

4.2.3 Experiences with the slow start

After analysing results of pingpong, we have seen that there is an important difference between the minimum and the maximum time to send one message. We use our pingpong to send 200 messages of 1 MB between one node on each site. Figure 9 represents the bandwidth for each of the 200 messages in function of time. On this figure, we can see the impact of TCP behavior (slowstart and congestion avoidance) on each MPI implementation. The slowstart and the congestion avoidance occurs on each node. TCP reaches the maximum bandwidth after 5s. GridMPI has the same behavior as TCP and the second phase is very short. However, in the other implementations the slowstart takes more time to reach the maximum value due to the second phase. In GridMPI, the value of 500 Mbps is reached in 2s and around 4s in the others. If we raised the message size to 16 MB, the maximum bandwith is obtained after 3 round-trips only on TCP and 6 round-trips on MPI implementations.

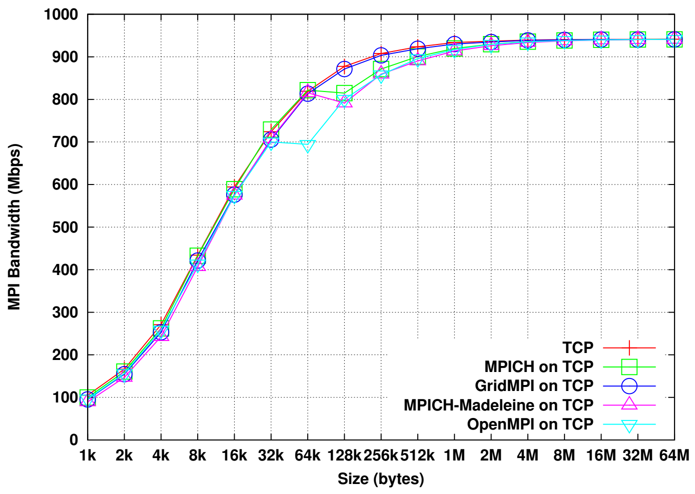


Figure 5: Comparison of MPI bandwidth of the different MPI implementations on a local network (cluster) with the default parameters

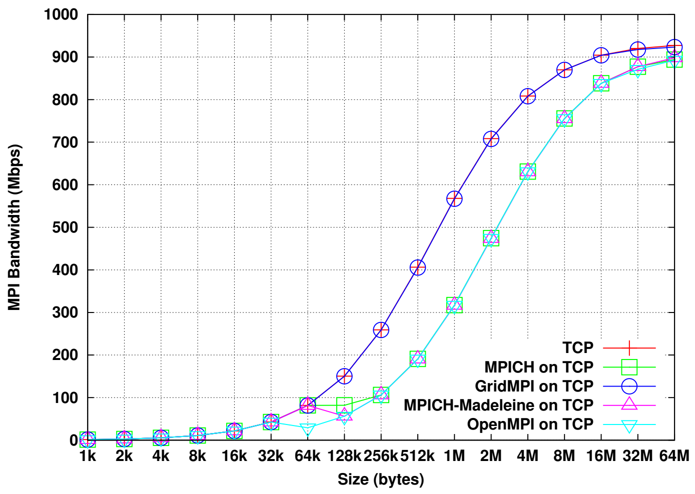


Figure 6: Comparison of MPI bandwith of the different MPI implementations on a distant network (grid) after TCP tuning

4.3 NAS Parallel Benchmarks

The two goals of the next experiments are to compare the performances of each implementation on the grid (Fig. 10 and to see if the grid can well execute MPI applications (Fig. 12 and 13).

In the next experiments, we execute the NPB 2.4. We used two different configurations: on the same cluster (using P_{R1} to P_{R16} see 2) or between two clusters joined by a WAN (using P_{R1} to P_{R8} and P_{N1} to P_{N8}) We execute each NPB 5 times and take the better time of these experiments.

Figure 10 shows the NAS execution time comparison of each MPI implementation. MPICH2 is taken as the reference implementation. The performance of the other implementations are relative to MPICH2.

As GridMPI optimize the collective operations, its speed-up is very important for the applications that communicate with collective operations (FT and IS). On MG, the performances are quite similar for each implementation excepted for MPICH2. However, the performances of MPICH2 are better on LU. Finally, execution time of BT and SP is only a little bit better with GridMPI. We can not obtained results with MPICH-Madeleine for BT and SP because the application timeout.

The next experiment aims to see which overhead the grid introduces compared to the execution on

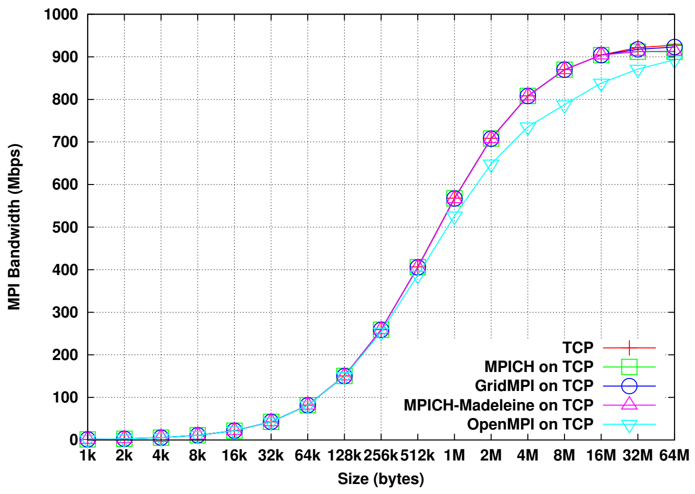


Figure 7: Comparison of MPI bandwidth of the different MPI implementations on a distant network (grid) after TCP tuning and MPI optimizations

a cluster. As Fig. 10 shows that GridMPI has a better behavior for each application, we choose this implementation to compare the NPB on the grid.

Figure 12 represents the relative performance of 16 nodes divided in two groups by a WAN to 16 nodes on the same cluster. This figure shows the impact of the latency for each kind of applications. EP has few communications thus the relative performance is close to 1. CG and MG bad performance because they communicate with little messages. The latency has a most important impact on little messages. LU's performance is good because of the message sizes. SP and BT communicate with big messages, thus the high latency does not impact them so much. The performance of IS is quite bad because GridMPI only optimizes one of the primitives used by IS (MPI_Allreduce). FT takes advantage of the optimization done on the MPI_Bcast primitive in GridMPI.

If we compare strictly the grid and the cluster with the same number of nodes, there is always an overhead. This overhead is less than 20% for half of the NAS. However, the main interest of the grids is to find resources that are not available on the cluster that's why Figure 13 represents the relative performance of 4 nodes in the same cluster to 16 nodes on two clusters.

The results for the performances of each application on the grid are quite similar of the last experiment. In this experiment, the speed-up should be equal to four if the grid had no impact. For CG and MG, even if we raise up the computing power four times, there is few speed-up. The long distance is very costly on applications with little messages. LU and BT have a very good speed-up near of 4 on the grid. FT and SP performances are quite good (the speed-up is at least 3).

These results confirm that it is efficient to execute an MPI application on the grid because there is a speed up for each application. However, applications with little messages have very bad performances due to high latency. Collective operations have to be optimized to be efficient.

4.4 Real application results

The physic application used is a software suite for seismology (see 2). It is composed of one master task and several slaves, 32 in our experiments. The master node sends a set of 1000 rays to the other nodes which compute them. When a slave node has finished computing, he asks for a new set of data. Thus, if a slave can compute faster than the others, it receives and computes more rays. Messages sent are quite small depending on the number of rays composing a set (69 kB for a set of 1000 rays).

When all the rays have been computed, the whole results have to be merged, in order to construct the final 3D mesh. Each node holds one submesh and has to merge all the information related to the cells of its submesh. First, each node sends the information it has computed to the recipient process. Secondly, it receives data computed by the others and related to his submesh.

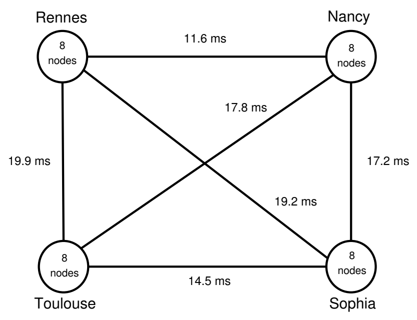


Figure 8: RTT times between each site for the execution of ray2mesh

During this phase, a huge amount of data is sent to each slave using the MPI_Isend point to point primitive. This phase generates an important trafic around 235 MB by node. This application is interesting because slaves do not receive the same number of rays, depending on the distance from the master they are and the CPU performance on which they are launched. The experiments with this application should help us to study the impact of tasks placement on the grid.

We execute ray2mesh on four different clusters (Nancy < Rennes, Toulouse < and Sophia, ordered by node capacity) with eight nodes on each as presented on Figure 8. We use a global set of 1 million of rays and we send groups of 1000 rays to slaves. We change the location of the master in the different clusters. Indeed, the nearer is a node from the master, the more it is receiving and computing data in the computing phase. If we put the master on the more powerful cluster we expect the performance to be better.

Table 6 shows the number of rays computed on each cluster depending of the master's location. Each node computes more rays if it is nearer of the master. The most powerful is a cluster, the most it computes rays.

Table 7 shows the mean of the computing, merging and total times obtained with the four locations of the master. The mean is done on 5 experiments. The total time is the sum of init, computing and merging time and also writing phase. These figures shows an equal performance during the computing phase not depending of the master's location. The merging phase times are a little bit different, due to network utilization during the experiments.

Table 6: Number of rays computed on each cluster depending of the master's location

		Master's location			
		Nancy	Rennes	Sophia	Toulouse
Cluster	Nancy	29650	27937,5	29343,75	28781,25
	Rennes	30225	30625	29437,5	29468,75
	Sophia	35375	36562,5	37343,75	36437,5
	Toulouse	29750	29875	28875	30312,5

5 Conclusion and Future Work

In this paper, we have presented a study of different MPI implementations in a grid context. We aimed at finding a good implementation to execute an application in this context. We also wanted to identify which application types and communication patterns are best suited to grids.

Our experiments are based on four MPI implementations (MPICH2, GridMPI, MPICH-Madeleine and OpenMPI). They show that TCP tuning and optimization in MPI implementation are necessary to obtain correct performance. After tuning, each MPI implementation can reach as good performance as TCP. To complete our study, we evaluate each implementation with the NPB. Results show that GridMPI behaves better than the others on Grid'5000. This is due to its optimizations of collective operations and of TCP. However, GridMPI can not manage heterogeneity. Execution of an application on a grid is efficient for

Table 7: Ray2mesh time results

Source	**Nancy**	**Rennes**	**Sophia**	**Toulouse**
Comp. time	185.11	185.16	186.03	186.97
Merge time	168.85	162.59	168.38	165.99
Total time	361.52	355.14	361.72	360.24

applications with quite big messages for point to point communications. Collective operations have to be improved to be efficient. Finally, we evaluate the behavior of a real grid application named ray2mesh. Results shows that the difference between execution time depends only of the network for this application, the task placement do not provide significantly better results.

In a near future, we will pursue our experiments with other Grid-aware implementations like MPICH-G2 and with other real MPI applications. Our results have shown problems with the slowstart and the congestion avoidance mechanisms of TCP when executing an application on the grid. We want to further explore these issues and study optimizations within TCP. Finally, we will test the heterogeneity management of each implementation with different high performance networks. Using these networks for local communications can be efficient to improve performance but has to remain simple. The overhead introduced by the management of heterogeneity has to be less important than the TCP cost.

Acknowledgment

This work has been funded by the French ministry of Education and Research via the EU IST FP6 EC-GIN Project. 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)

References

[1] Guillaume Aumage, Olivier et Mercier. MPICH/MADIII : a Cluster of Clusters Enabled MPI Implementation. In *Proceedings of CCGrid*, Tokyo, Japan, 2003.

[2] Olivier Aumage. Heterogeneous multi-cluster networking with the Madeleine III communication library. In *16th International Parallel and Distributed Processing Symposium*, Florida, USA, 2002.

[3] Olivier Aumage, Elisabeth Brunet, Nathalie Furmento, and Raymond Namyst. Newmadeleine: a fast communication scheduling engine for high performance networks. In *CAC 2007: Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2007*, Long Beach, California, USA, March 2007. Also available as LaBRI Report 1421-07 and INRIA RR-6085.

[4] D Bailey, E Barscz, J Barton, D Browning, R Carter, L Dagum, R Fatoohi, S Fineberg, P Frederickson, T Lasinski, R Schreiber, H Simon, V Venkatakrishnan, and S Weeratunga. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, California, USA, 1994.

[5] H Benoit-Cattin, F Bellet, J Montagnat, and C Odet. Magnetic Resonance Imaging (MRI) simulation on a grid computing architecture. In *Proceedings of IEEE CGIGRID'03-BIOGRID'03*, pages 582–587, Tokyo, 2003.

[6] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lantéri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Touche Iréa. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006. https://www.grid5000.fr/.

[7] George Bosilca, Aurélien Bouteiller, Franck Cappello, Samir Djilali, Gilles Fédak, Cécile Germain, Thomas Hérault, Pierre Lemarinier, Oleg Lodygensky, Frédéric Magniette, Vincent Néri, and Anton Selikhov. MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. In *Proceedings of The IEEE/ACM SC2002 Conference*, 2002.

[8] Darius Buntinas, Guillaume Mercier, and William Gropp. Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem. Technical Report P1346, Mathematics and Computer Science Division, Argonne National Laboratory, Chicago, USA, 2006.

[9] Thomas Peiselt Dietmar Fey Christian Kauhaus, Adrian Knoth. Efficient message passing on multi-clusters: An ipv6 extension to Open MPI. In *Proceedings of KiCC'07, Chemnitzer Informatik Berichte*, February 2007.

[10] Graham E. Fagg and Jack Dongarra. Ft-mpi: Fault tolerant mpi, supporting dynamic applications in a dynamic world. In *PVM/MPI*, pages 346–353, 2000.

[11] Ahmad Faraj and Xin Yuan. Communication Characteristics in the NAS Parallel Benchmarks. In *IASTED PDCS*, pages 724–729, 2002.

[12] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[13] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[14] Stéphane Genaud, Marc Grunberg, and Catherine Mongenet. Experiments in running a scientific MPI application on Grid'5000. In *4th High Performance Grid Computing International Workshop, IPDPS conference proceedings*, Long beach, USA, March 2007. IEEE.

[15] GridMPI Project. `http://www.gridmpi.org/gridmpi.jsp`.

[16] William Gropp. MPICH2: A New Start for MPI Implementations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 9th European PVM/MPI Users' Group Meeting*, Linz, Austria, October 2002.

[17] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[18] Marc Grunberg, Stéphane Genaud, and Catherine Mongenet. Seismic ray-tracing and Earth mesh modeling on various parallel architectures. *The Journal of Supercomputing*, 29(1):27–44, July 2004.

[19] Matt Mathis and Raghu Reddy. Enabling High Performance Data Transfers on Hosts. Technical report, Pittsburgh Supercomputer Center, 1999. `http://www.psc.edu/networking/projects/tcptune/`.

[20] M Matsuda, T Kudoh, Y Kodama, R Takano, and Y Ishikawa. TCP Adaptation for MPI on Long-and-Fat Networks. In *Proceedings of Cluster05*, Boston, Massachusetts, USA, September 2005.

[21] M Matsuda, T Kudoh, Y Kodama, R Takano, and Y Ishikawa. Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks. In *Proceedings of Cluster06*, Barcelona, Spain, September 2006. `http://www.gridmpi.org/publications/cluster06-matsuda.pdf`.

[22] M Matsuda, T Kudoh, H Tazuka, and Y Ishikawa. The design and implementation of an asynchronous communication mechanism for the MPI communication model. In *Proceedings of Cluster04*, San Diego, California, USA, September 2004.

[23] Matthias Müller, Matthias Hess, and Edgar Gabriel. Grid enabled MPI solutions for Clusters. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 18, Washington, DC, USA, 2003. IEEE Computer Society.

[24] I. Foster N. Karonis, B. Toonen. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, pages 551–563, 2003.

[25] Scott Pakin and Avneesh Pant. Vmi 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management. In *Proceedings of HPCA-8*, 2002.

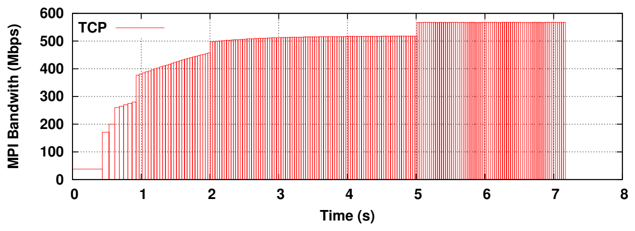
[26] A. Pant and H. Jafri. Communicating efficiently on cluster based grids with MPICH-VMI. In *Proceedings of Cluster Computing*, 2004.

[27] Site RENATER. Renater4. `http://www.renater.fr/`.

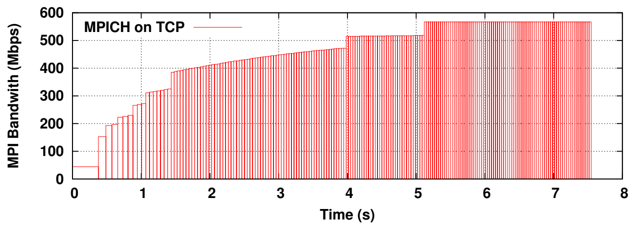
[28] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis. Automatic TCP buffer tuning. In *Proceedings of the ACM SIGCOMM '98*, pages 315–323, New York, NY, USA, 1998. ACM Press.

[29] Jaspal Subhlok, Shreenivasa Venkataramaiah, and Amitoj Singh. Characterizing NAS Benchmark Performance on Shared Heterogeneous Networks. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 91, Washington, DC, USA, 2002. IEEE Computer Society.

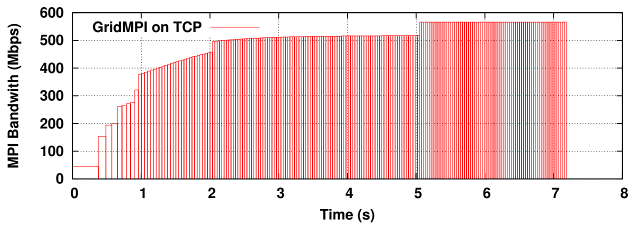
[30] R Takano, T Kudoh, Y Kodama, M Matsuda, H Tezuka, and Y Ishikawa. Design and evaluation of precise software pacing mechanisms for fast long-distance networks. In *Proceedings of PFLDnet05*, Lyon, France, February 2005.



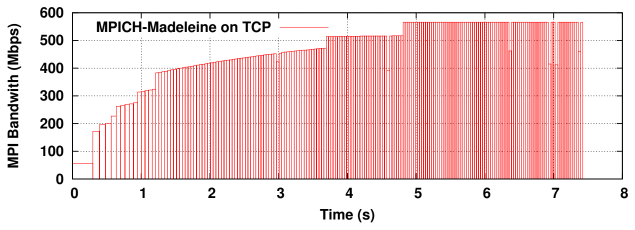
(a) Impact of the slowstart on TCP



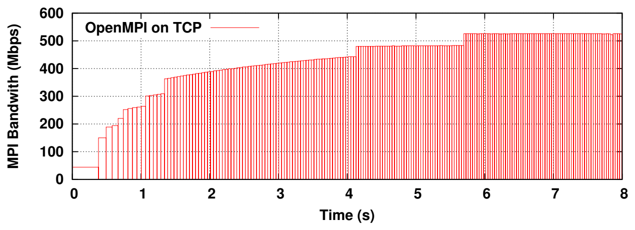
(b) Impact of the slowstart on MPICH



(c) Impact of the slowstart on GridMPI



(d) Impact of the slowstart on MPICH-Madeleine



(e) Impact of the slowstart on OpenMPI

Figure 9: Impact of the TCP slowstart on the gird with MPI implementations

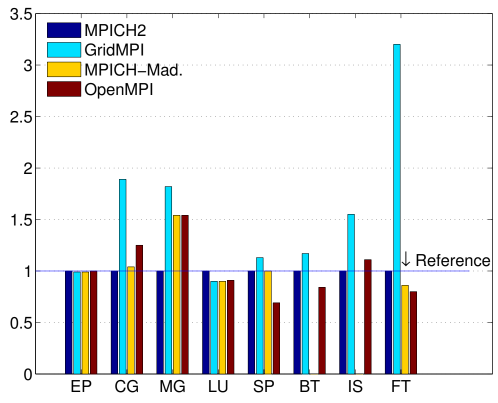


Figure 10: Relative comparison between the different MPI implementations on 8-8 nodes between two clusters, Ref. is MPICH2

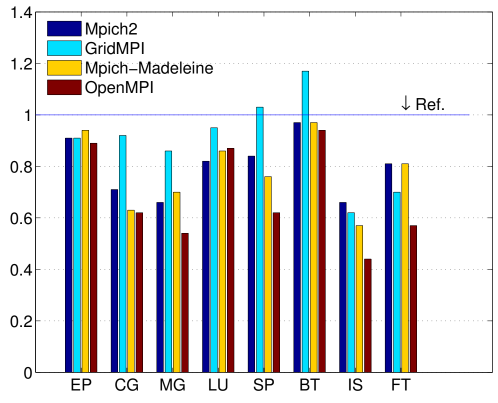


Figure 11: Relative comparison between the different MPI implementations on 2-2 nodes between two clusters, Ref. is MPICH2

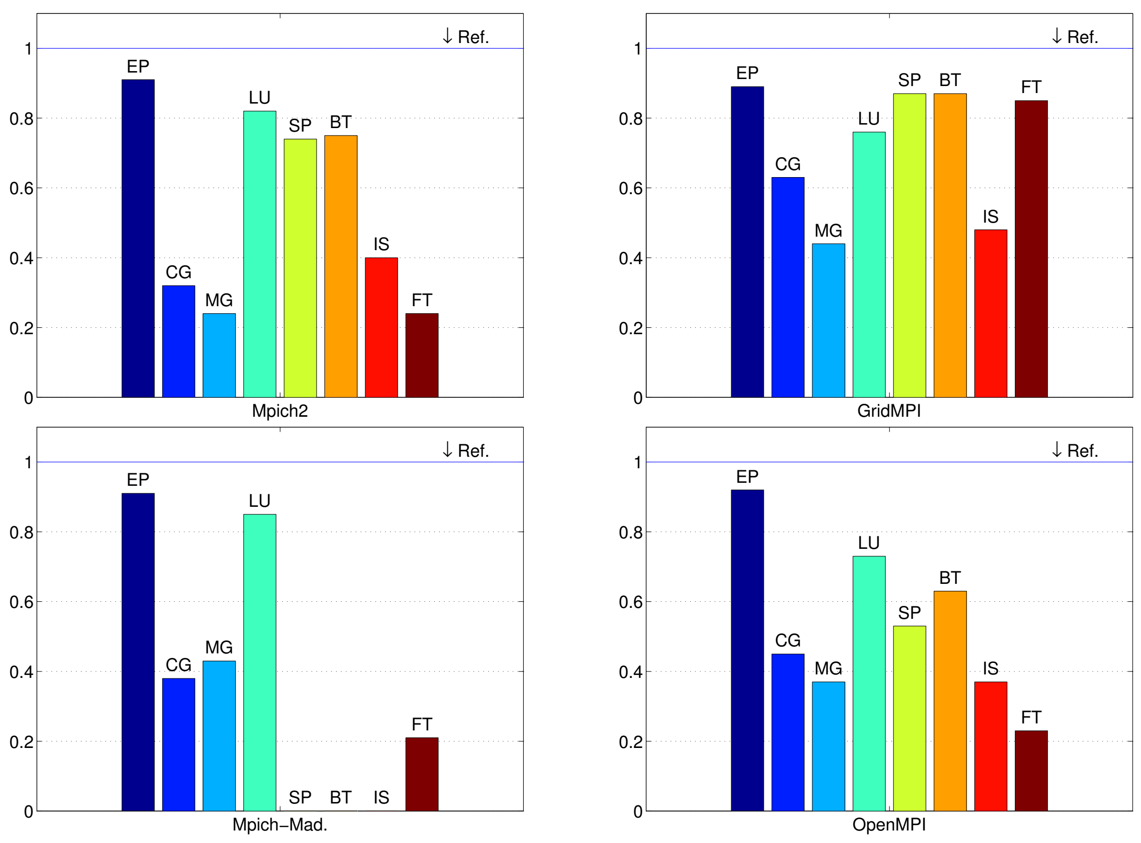


Figure 12: Relative comparison for each implementation between 16 nodes on the same cluster and 8-8 nodes on 2 different clusters, Ref. is 16 nodes on the same cluster

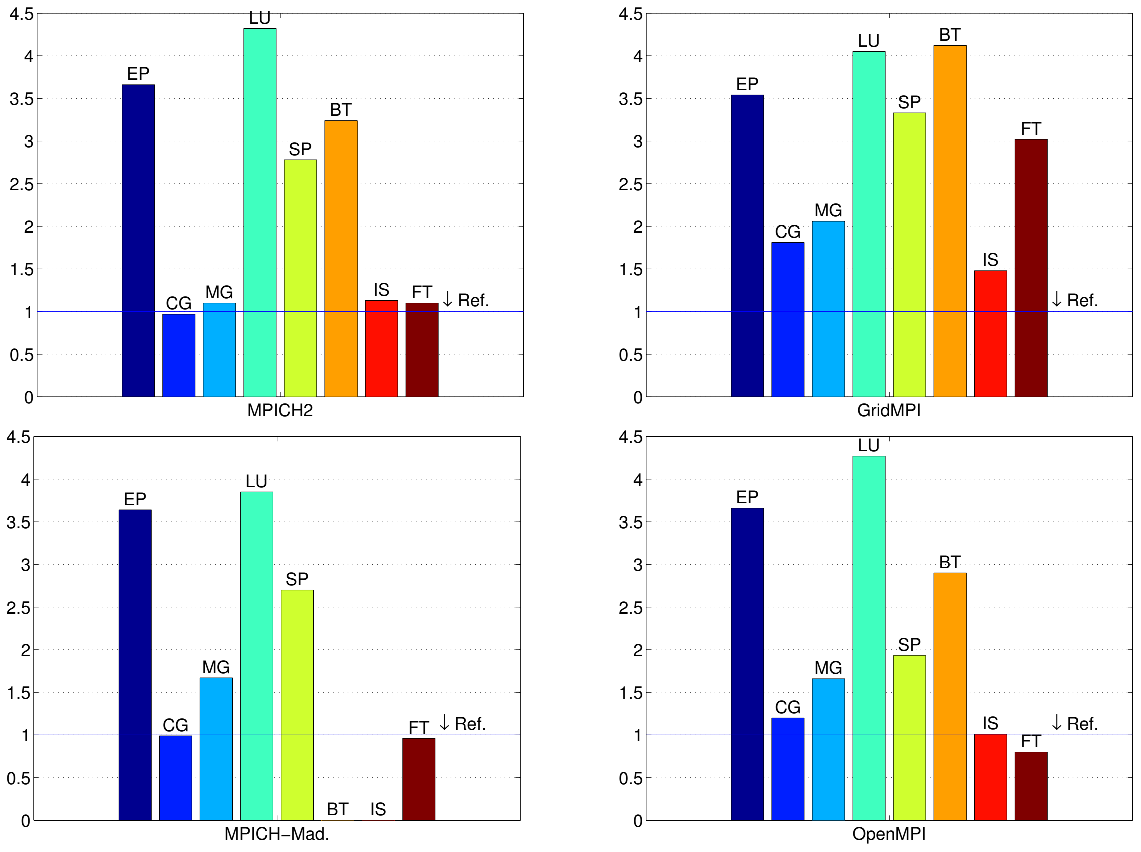


Figure 13: Relative comparison for each implementation between 4 nodes on the same cluster and 8-8 nodes on two clusters, Ref. is 4 nodes on the same cluster