Linear Algebra Computation Benchmarks on a Model Grid Platform

Loriano Storchi[1], Carlo Manuali[2], Osvaldo Gervasi[3], Giuseppe Vitillaro[4], Antonio Laganà[1], and Francesco Tarantelli[1,4]

[1] Department of Chemistry, University of Perugia,
via Elce di Sotto, 8, I-06123 Perugia, Italy
redo@thch.unipg.it,lag@unipg.it,franc@thch.unipg.it

[2] CASI, University of Perugia,
via G. Duranti 1/A, I-06125 Perugia, Italy
carlo@unipg.it

[3] Department of Mathematics and Informatics, University of Perugia,
via Vanvitelli, 1, I-06123 Perugia, Italy
osvaldo@unipg.it

[4] Istituto di Scienze e Tecnologie Molecolari, CNR,
via Elce di Sotto, 8, I-06123 Perugia, Italy
peppe@thch.unipg.it

Abstract. The interest of the scientific community in Beowulf clusters and Grid computing infrastructures is continuously increasing. The present work reports on a customization of Globus Software Toolkit 2 for a Grid infrastructure based on Beowulf clusters, aimed at analyzing and optimizing its performance. We illustrate the platform topology and the strategy we adopted to implement the various levels of process communication based on Globus and MPI. Communication benchmarks and computational tests based on parallel linear algebra routines widely used in computational chemistry applications have been carried out on a model Grid infrastructure composed of three 3 Beowulf clusters connected through an ATM WAN (16 Mbps).

1 Introduction

Grid computing [1] is promising to establish itself as a revolutionary approach to the use of computing resources. At the same time, Beowulf-type clusters (BC) [2] have become very popular as computing platforms for the academic and scientific communities, showing extraordinary stability and fault tolerance at very attractive cost/benefits ratios. The availability of Gigabit Ethernet further facilitates the assembly of high throughput workstation clusters of the Beowulf type. It is therefore a natural and smooth development to explore the possibility of integration between the Grid and the cluster paradigms. By connecting together, at the hardware and software levels, several clusters one can in principle build a Grid computing platform very useful for scientific applications. While the Grid model is often viewed as a cooperative collection of individual computers, it is clear that, in order to make such a model efficient for scientific applications, and we think in particular of computational chemistry ones, much effort must be devoted toward re-designing the scheduling and communication software (and possibly the application themselves) so that the Grid topology and interconnections are explicitly taken into account. As we mentioned, this refers especially to local clustering. Some work along these lines was for example carried out by developing a MPI-based library of collective communication operations explicitly designed to take into account two layers (wide-area and local-area) of message passing [3].

The present work presents a study of the performance of a model platform composed of three Beowulf clusters connected via Internet and assembled as a Grid based on Globus Toolkit 2 [4], with a view on using it for quantum chemistry applications. The aspects of the computational environment and the optimizations we have focused on concern in particular:

1. a centralized installation of the Globus software into a NFS shared directory and the definition of a method to specify the cluster's hosts parameters for MDS.
2. An implementation of Globus and MPICH-G2 [5] for Grid management taking explicitly into account the local tightly-coupled MPI level (LAM/MPI in our case) for the intra-cluster communication among the nodes.
3. A modification of the LAM/MPI *broadcast* implementation in order to optimize LAM/MPI throughput on the Grid.

The resulting computing model shows very promising features and supports the existing wide interest in the Grid approach to computing. In Sec. 2 we give some details of the platform we have set up. In Sections 3 and 4 we illustrate the need for *topology-awareness* by discussing the performance of some broadcast schemes. In Sec. 5 the performance of some parallel linear algebra kernels very important for computational chemistry applications is analyzed.

2 A Model Computing Grid

The work described in the present paper has been carried out on a computing platform composed of three Beowulf-type workstation clusters, named *GRID*, *HPC* and *GIZA*. The nodes within each cluster are interconnected through a dedicated switched network, with bandwidth of 100 Mbps for *HPC* (Fast-Ethernet), 200 Mbps for *GIZA* (Fast-Ethernet with 2 NICs in Channel Bonding [6]) and 1 Gbps (Gigabit-Ethernet) for *GRID*. *HPC* and *GRID* are interconnected through a Fast-Ethernet local area network (LAN) at 100 Mbps, while they are connected to *GIZA* over a 16Mbps ATM wide area network (WAN). The connection among the three clusters is schematically shown in Fig. 1, where the the effective average communication bandwidth is indicated. Table 1 summarizes the main characteristics of each cluster.

On our model Grid we have installed Globus Toolkit 2, introducing some modifications of the reference installation procedure [4] which are of general relevance for Grids based on Beowulf clusters. In a Beowulf cluster environment it is common practice to adopt a configuration where a specialized node, called

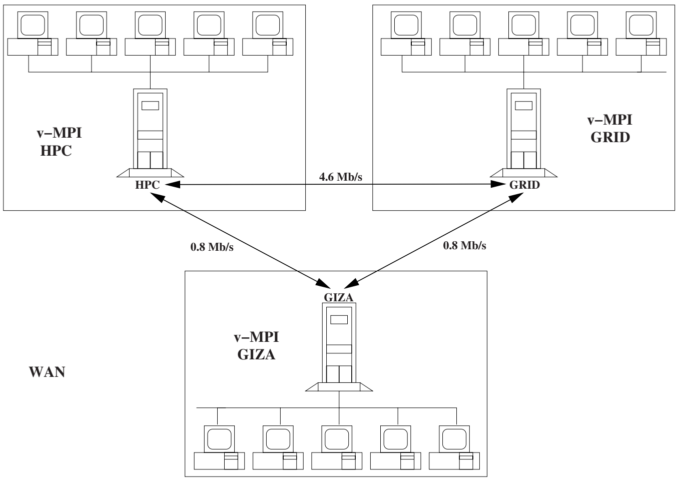

Fig. 1. Representation of the model Grid.

frontend, acts as a firewall for the other nodes and hosts all the Internet services needed by the cluster. In particular, this node usually hosts the NIS to allow one-time login of users, the shared (NFS exported) filesystems and the Automount service. In particular, the directory `/usr/local` is exported via NFS and contains the software packages shared by the other cluster nodes. In our case, all nodes access Globus in `/usr/local/globus`. For security reasons, we have restricted the range of port numbers Globus may use. The information about the nodes used by MDS is kept in the custom directory `/usr/local/globus/etc/nodes`, where a subdirectory for each node of the cluster is created to keep the GRIS node configuration files. This directory must be named as the value of the `HOSTNAME` environment variable. On each node, the GRIS subsystem is activated at boot time, invoking the `SXXgris` command customized in order to point to the right configuration directory defined by the environment variable `sysconfdir`. In this way all nodes refer to the GIIS server running on the frontend of the cluster and all the resources of the cluster may be inspected trough the LDAP server. To implement such configuration, some modifications to the MDS configuration files are necessary. In particular, in each node the following line must appear in `Grid-info-site-policy.conf` so that all MDS operations from any node of the cluster are accepted:

`policydata: (&(Mds-Service-hn=*.`*cluster-IP-domain*`)(Mds-Service-port=2135))`

and the file `Grid-info-resource-register.conf` in each node of the cluster, in order to register the local GRIS server on the GIIS server, must contain:

`reghn:` *GIIS-server*. *cluster-IP-domain*
`hn:` *GRIS-server*. *cluster-IP-domain*

Table 1. Characteristics of the three Beowulf clusters used for the model Grid.

Cluster:	*GRID*	*HPC*	*GIZA*
Number of nodes:	9 SMP working nodes, 18 working CPU	8 SMP working nodes, 16 working CPU	8 SMP working nodes
Processor type:	Intel Pentium III double-processor	Intel Pentium II double-processor	Intel Pentium III single-processor
Clock:	18 CPUs at 1GHz	4 CPUs at 550 Mhz, 1 CPU at 450 Mhz, 3 CPUs at 400 Mhz	8 CPUs at 800 Mhz
RAM:	18 Gbyte	4 Gbyte	4 Gbyte
HDISK:	EIDE/SCSI	EIDE/SCSI	EIDE
Type of switched Network:	Gigabit-Ethernet	Fast-Ethernet	Fast-Ethernet, 2 channels bonded
Linux Kernel type:	RedHat 7.2 2.4.13 SMP MOSIX	RedHat 6.2 2.2.16-3 SMP	RedHat 7.0 2.2.19

One very important aspect of our exercise was that we wanted to retain, in the Globus Grid, the ability to exploit the local level of MPI parallelism on each cluster in a very general way. As the local level of MPI environment, we have chosen to adopt LAM/MPI (version 6.5.6). To use the cluster's local MPI implementation one needs to compile the *Globus Resource Management* SDK with the flavor `mpi`. As has already been noted [7], this operation fails on Beowulf nodes (only some vendor MPI are supported). To overcome this problem, we have recompiled the package `globus_core` according to the following sequence of commands:

```
# export GLOBUS_CC=gcc;
# export LDFLAGS='-L/usr/local/globus/lib'
# /usr/local/globus/BUILD/globus_core-2.1/configure
  --with-flavor=gcc32mpi --enable-debug --with-mpi
  --with-mpi-includes=-I/usr/include/
  --with-mpi-libs="-L/usr/lib -lmpi -llam"
# make all
# make install
```

After successful compilation of `globus_core`, the installation of the *Globus Resource Management* SDK can be accomplished without problems. In order to be able to build the MPICH-G2 package correctly, it is further necessary to introduce some defines which are missing in the include file `mpi.h` of LAM/MPI.

```
#define MPI_CHARACTER         ((MPI_Datatype) &lam_mpi_character)
#define MPI_COMPLEX           ((MPI_Datatype) &lam_mpi_cplex)
#define MPI_DOUBLE_COMPLEX    ((MPI_Datatype) &lam_mpi_dblcplex)
#define MPI_LOGICAL           ((MPI_Datatype) &lam_mpi_logic)
#define MPI_REAL              ((MPI_Datatype) &lam_mpi_real)
#define MPI_DOUBLE_PRECISION  ((MPI_Datatype) &lam_mpi_dblprec)
#define MPI_INTEGER           ((MPI_Datatype) &lam_mpi_integer)
#define MPI_2INTEGER          ((MPI_Datatype) &lam_mpi_2integer)
#define MPI_2REAL             ((MPI_Datatype) &lam_mpi_2real)
#define MPI_2DOUBLE_PRECISION ((MPI_Datatype) &lam_mpi_2dblprec)
```

To enable the spawning of MPICH-G2 jobs and to simplify the addition of nodes to the cluster we let the `$GLOBUS_GRAM_JOB_MANAGER_MPIRUN` macro point to the following script replacing the standard `mpirun` command:

```
export LAMRSH=rsh
export LAM_MPI_SOCKET_SUFFIX="GJOB"$$
/usr/bin/lamboot /usr/local/globus/hosts >> /dev/null 2>&1
/usr/bin/mpirun -c2c -O -x '/usr/local/globus/bin/glob_env' $*
rc=$?
/usr/bin/lamhalt >> /dev/null 2>&1
exit $rc
```

Here `glob_env` is a small program which returns the list of current environment variables so that they can be exported by `mpirun` (`-x` flag).

3 Topology-Aware Functions: Broadcast Models

Since a Grid is made up of many individual machines connected in a Wide Area Network (WAN), two generic processes may be connected by links of different kind, resulting in widely varying point-to-point communication performance. Therefore, the availability of topology-aware collective functionalities, and more generally of topology discovery mechanisms, appears to be of fundamental importance for the optimization of communication and performance over the Grid. The aim of such topology-aware strategies would be to minimize point-to-point communications over slow links in favor of that over high bandwidth and/or low latency links. In some cases it may also be possible to schedule data exchange so that as much slow-bandwidth communication as possible is hidden behind fast traffic and process activity. Knowledge of the topology of a Grid is ultimately knowledge of the characteristics of the communication link between any two processes at any given time. A first useful approximation to such detailed map is provided by the classification scheme described in the MPICH-G2 specifications [5, 8]. In this scheme there are four levels of communication: levels 1 and 2 account for TCP/IP communications over a WAN and a LAN, respectively. Level 3 concerns processes running on the same machine and communicating via TCP/IP, while level 4 entails communication through the methods provided by the local MPI implementations. Thus, the MPICH-G2 layer implements topology awareness over levels 1, 2 and 3, while level 4 operations are left to the vendor-supplied implementation of MPI.

We have tried to obtain a first demonstration of the benefits of topology-aware collectives by comparing the performance on our Grid of three different broadcast methods: *(i)* the broadcast operation provided by MPICH-G2, *(ii)* an optimized topology-aware broadcast of our own implementation, and *(iii)* a non-topology-aware broadcast based on a flat binomial-tree algorithm. As is clear, all processes can communicate at Level 1, because all machines are in the same WAN, but only processes that run on machines belonging to the same cluster (*HPC*, *GRID* or *GIZA*) can communicate using the locally supplied MPI (l-MPI) methods. The l-MPI we have used is LAM/MPI [9] and provides for communication via TCP/IP among nodes in a dedicated network or via shared-memory for processes running on the same machine. In accord with the MPICH-G2 communication hierarchy, we can thus essentially distinguish between two point-to-point communication levels: inter-cluster communication (Level 1) and intra-cluster communication (Level 4). As already mentioned, we notice that the effective bandwidth connecting *GIZA* to *HPC* and *GRID* is slower than that between *HPC* and *GRID*. This asymmetry may be thought of as simulating the communication inhomogeneity of a general Grid.

Consider now a typical broadcast, where one has to propagate some data to 24 machines, 8 in each cluster. For convenience, the machines in cluster *HPC* will be denoted $p_0, p_1, \ldots, p_7$, those in *GRID* as $p_8, p_9, \ldots, p_{15}$, and those in *GIZA* as $p_{16}, p_{17}, \ldots, p_{23}$. The MPICH-G2 `MPI_Bcast` operation over the communicator `MPI_COMM_WORLD`, rooted at p_0, produces a cascade of broadcasts, one for each communication level. Thus, in this case, there will be a broadcast at the WAN inter-cluster level, involving processes p_0, $p8$ and p_{16}, followed by three intra-cluster propagations, where l-MPI will take over. So we have just two inter-cluster point-to-point communication steps, one from p_0 to p_8 and another from p_0 to p_{16}, and then a number of intra-cluster communications. The crucial point to be made here is that communication over the slow links is minimized, while the three fast local (intra-cluster) broadcast propagations can take place in parallel.

In this prototype situation, the strategy adopted in our own implementation of the broadcast is essentially identical, but we have optimized the broadcast operation at the local level. The essential difference between our implementation of the broadcast and the LAM/MPI one is that in the latter, when a node propagates its data via TCP/IP, non-blocking (asynchronous) send operations over simultaneously open sockets are issued, while we opted for blocking operations. The local broadcast tree is depicted in Fig. 2. The LAM/MPI choice appears to be optimal on a high bandwidth network where each node is connected inde-

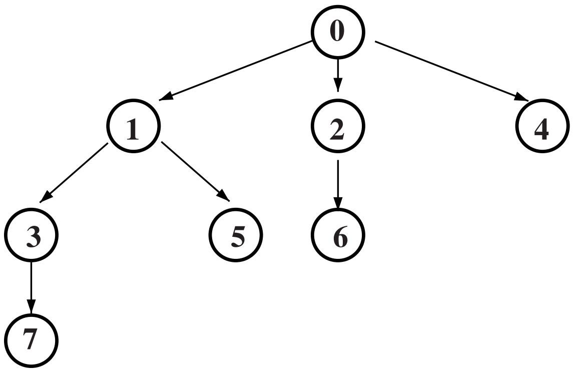

Fig. 2. Local broadcast tree in a 8-node cluster.

pendently to all the others, but it clearly loses efficiency on a switched-ethernet cluster, where simultaneous send operations issued by a node necessarily share the available bandwidth. It is not difficult to see that on small clusters this translates in a remarkable efficiency loss. In particular, in our 8-node case, with the tree of Fig. 2, a factor of two in broadcast time is observed: if τ is the basic transmission *time step*, i.e., the time required for data exchange between any two nodes in an isolated send/receive operation (for large data transfers this is roughly the amount of data divided by the bandwidth), then the synchronous broadcast completes in about 6τ, while our version takes just 3τ.

4 Broadcast Tests

In all our tests we have propagated ~38 Mb of data ($5 \cdot 10^6$ double precision real numbers). Thus, on a local switched 100 Mbit ethernet such as that of the *HPC* cluster the measured time-step is $\tau = 3.4$ s and the optimized broadcast takes about 10 seconds. On *GIZA*, where each node multiplexes over two ethernet cards, the time is exactly halved. It is instructive to compare the performance of topology-aware broadcasts with the no-topology-aware one. The latter, as previously mentioned, is executed using a flat binomial tree algorithm involving all nodes of the Grid, without consideration for the different link speeds. This procedure of course results in a larger amount of slow inter-cluster communication at the expenses of the fast local transfers. An example is shown in Fig. 3 where a broadcast is propagated again from node p_0. As can be seen, we have here only 7

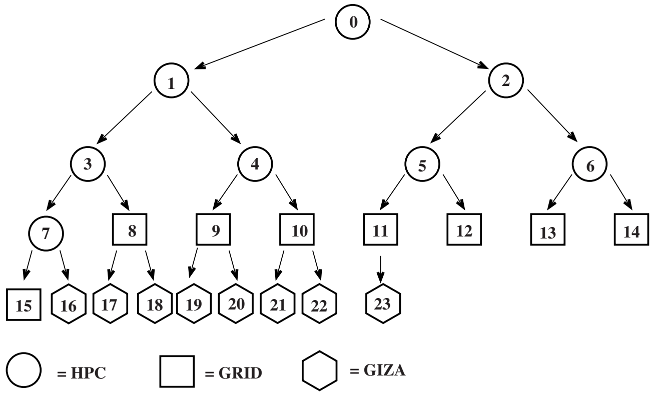

Fig. 3. Flat broadcast tree over the whole Grid.

intra-cluster data transfers (the same number we actually have in a single local broadcast) and as many as 16 inter-cluster data transfers instead of the two we have in the topology-aware algorithm. By taking into account the various link speeds it is possible to give a rough estimate of the overall times required for the three broadcast algorithms. For a 38Mb data propagation, as before, these are compared with the measured ones in Table 2. It should be noted that the

Table 2. Performance of different broadcast algorithms over the Grid.

Broadcast type	**Measured time** (s)	**Estimated time** (s)
MPICH-G2	70	69
Locally optimized	60	59
No-topology aware	448	440

serial link between either *HPC* or *GRID* and *GIZA* is not dedicated. Thus, the reported link speed of 0.8 Mbyte/s (see Fig. 1) is an average value obtained over various measurements at the time of our study. In the first case of Table 2 the broadcast operation is executed by MPICH-G2 in two steps. First, a broadcast over the WAN takes place, consisting in practice of two non-blocking sends from p_0 (*HPC*) to p_8 (*GRID*) and p_{16} (*GIZA*). The rate of these data transfers falls much below the available local bandwidth and they go through different routes, thus they should overlap quite effectively: the overall time required coincides with the time of the *HPC*-*GIZA* transfer, i.e. about 49 s. After the inter-cluster broadcast, three parallel local broadcasts, one on each cluster. The time for these operations is the largest of the three, i.e. ∼20 seconds, required on *HPC* (LAM/MPI is used). Thus, the total time for a MPICH-G2 global broadcast is estimated in 69 seconds, matching closely the observed time. The second case of Table 2 differs from the first only for the optimization of the local intra-cluster broadcasts. As we have shown, this halves the local time from 20 to 10 seconds.

The last case of broadcast reported in Table 2 is the flat no-topology-aware structure whose tree is sketched in Fig. 3. All data transfers originating from the same node are synchronous (blocking). First of all, a local broadcast within *HPC* takes place. After 3 time steps (∼10 s) all local nodes except p_6 have been reached and p_3, p_4 and p_5 simultaneously start their sends toward *GRID*. The time for these would be ∼32 s, but it is in fact longer because after the first 3.4 s also p_6 starts sending through the same channel toward *GRID*. A better estimate yields 38 s. After this the 8 transfers from *HPC* and *GRID* toward *GIZA* start taking place and the time required for these, sharing the same channel, is of course of the order of 400 s. In total, the observed broadcast time of 448 seconds is therefore very well explained and the dominance of the long distance transfers, which in a topology-aware implementation is suitably minimized, is evident.

5 Linear Algebra Benchmarks

Since the intended use of our cluster Grid is for quantum chemistry and nuclear dynamics applications, we have also performed some preliminary tests on its performance with standard linear algebra routines. To this purpose, we have installed the ScaLAPACK package [10] on top of MPICH-G2. The resulting software hierarchy is depicted in Fig. 4. As the figure shows, we have installed, besides the standard BLAS and LAPACK software packages at the local level, the BLACS software (Basic Linear Algebra Communication Subprograms) [11] on top of MPICH-G2, PBLAS (a parallel BLAS library) and ScaLAPACK, a

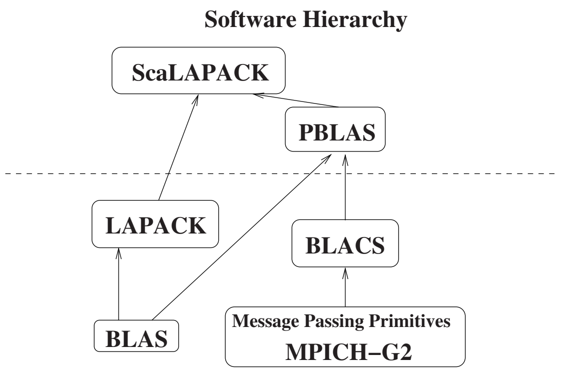

Fig. 4. ScaLAPACK software hierarchy on the Grid. The broken line separates the local environment (below) from the global one (above).

library of high-performance parallel linear algebra routines. BLACS is a message-passing library designed for linear algebra and it implements a data distribution model consisting of a one- or two-dimensional array of processes, each process storing a portion of a given array in block-cyclic decomposition. On this data distribution scheme PBLAS and ScaLAPACK work. Besides parallel execution, the data distribution model enables the handling of much larger arrays than would be possible in a data replication scheme.

The initial tests we have made use one of the fundamental PBLAS routines, PDGEMM, which performs matrix multiplication on double precision floating point arrays. The algorithm used in PDGEMM is essentially the one described in ref. [12] where a two-dimensional block-cyclic decomposition of the arrays is distributed and data transfers occur mostly along one dimension. Thus, our Grid of workstation clusters lends itself quite naturally to such decomposition if we arrange the machines in each cluster along a different row (say) of the BLACS array and ensure that data communication (the exchange of array blocks) takes place predominantly along rows of the array (i.e., intra-cluster).

We have measured PDGEMM performance for such an arrangement for the multiplication of two 20000 by 20000 matrices observing an effective speed of about 2.5 Gflops. Comparing this to the theoretical aggregate speed of the Grid in the absence of communication, it is clear that data transfers dominate the Grid activity. This is confirmed also by the fact that the global performance is almost completely independent of the size of the array blocks used in the block-cyclic decomposition: the measured speed varies from 2.52 Gflops to 2.55 Gflops for block sizes 64 to 256 (but the top speed is reached already for a block size of 160). The block size substantially affects local matrix multiply performance through the extent of cache re-use and a larger block size probably also improves data communication slightly by decreasing the impact of network latency. It should finally be noted that by exchanging rows with columns of the processor array, i.e., increasing the amount of inter-cluster data transfers, the expected performance deterioration reaches 70%.

6 Conclusions

In this work we have discussed how to realize, on a model Grid made up of three workstation cluster, two Globus communication levels, inter-cluster and intra-cluster, using MPI. The benefits of topology-aware communication mechanisms have been demonstrated by comparing a two-level implementation of broadcast with a flat binary-tree one. The model Grid has been further tested by measuring the performance of parallel linear algebra kernels of common use in theoretical chemistry applications, arranged to exploit the two communication levels. The tests have provided useful indications on how to make efficient use of a Grid platform composed of workstation clusters.

Acknowledgments

This research has been financially supported by MIUR, ASI and CNR.

References

1. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, USA (1999)
2. See e.g.:`http://www.beowulf.org`
3. Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.A.F.: MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems. ACM Sigplan Notices, Vol. 34 (1999) 131–140
4. The Globus Project:`http://www.globus.org`
5. Foster, I., Karonis, N.: A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. SC'98, Orlando, Florida (1998). See also `http://www3.niu.edu/mpi/`
6. See e.g.:`http://sourceforge.net/projects/bonding/`
7. See the Globus mailing lists, e.g.: `http://www-unix.globus.org/mail_archive/mpich-g/2002/04/msg00007.html`
8. Karonis, N., de Supinski, B., Foster, I., Gropp, W., Lusk, E., Bresnahan, J.: Exploiting hierarchy in parallel computer networks to optimize collective operation performance. Fourteenth International Parallel and Distributed Processing Symposium, Cancun, Mexico (2000).
9. `http://www.lam-mpi.org/`
10. `http://www.netlib.org/scalapack/slug/scalapack_slug.html`
11. (a) Dongarra, J., Van de Geijn, R.: Two dimensional basic linear algebra communication subprograms. Computer Science Dept. Technical Report CS-91-138, University of Tennessee, Knoxville, USA (1991) (also LAPACK Working Note #37); (b) Dongarra, J., Van de Geijn, R., Walker, D.: Scalability issues in the design of a library for dense linear algebra. Journal of Parallel and Distributed Computing, Vol. 22, N. 3 (1994) 523-537 (also LAPACK Working Note #43).
12. Fox, G., Otto, S., Hey, A.: Matrix algorithm on a hypercube I: matrix multiplication. Parallel Computing, Vol. 3 (1987) 17–31