



Workgroup 1: Resource Integration and Grid Support

MPI over the Grid¹

Deliverable	1.5 AstroGrid-D MPI over the Grid
Authors	Workgroup 1: Resource Integration and Grid Support
Editors	H. Enke, S. White
Date	10-04-2008
Document Version	1.0.0
Current Version	1.0.0
Previous Versions	0.2.1,0.2.0,0.1.0

A: Status of this Document

Published Deliverable 1.5, Version 1.0.0, describing options and basic procedures for installation and use of software implementing the Message Passing Protocol (MPI) over the Grid.

B: Reference to project plan

In this document an ansatz for computing on distributed cluster is studied. Cosmological simulations often show relations between the amount of available CPU and the amount of particles in the simulation, or more generally the size of the simulation and the characteristics of available hardware. Coupling clusters could be a way to overcome this scaling problem.

¹This work is part of the AstroGrid-D project and D-Grid. The project is funded by the German Federal Ministry of Education and Research (BMBF).

C: Abstract

This document describes the installation of MPI with Globus, and how to use it to pass information between processes running on remote Grid resources.

D: Changes History

Version	Date	Name	Brief summary
0.1.0	13.12.2007	S. White	Working Draft Creation
0.2.0	04.02.2008	S. White	Announcement of completion
0.2.1	04.02.2008	S. White	Revised deliverable number
0.2.2	18.03.2008	S. White, H. Enke	Revised text, added subsect. 5.3, minor text edits
1.0.0	10.04.2008	S. White, H. Enke	Final revision after approval period, no comments and change suggestions received

E:

Contents

1	Introduction to MPI-over-Grid	5
2	Existing software	5
3	Default Procedures for MPICH-G2	7
4	Installation of MPICH-G2	7
4.1	Required software	7
4.2	Preparation	7
4.3	Configure	8
4.4	Build and Install	8
5	Build an MPI test program	9
5.1	Example on a Linux system	9
5.2	Example on an IBM AIX/POWER system	9
5.3	Example with mpicc	10
6	Run an MPI program over the Grid	10
6.1	The machines file	11
6.2	Globus modules involved	11
6.3	Runs on heterogeneous grids with RSL scripts	11
6.4	File staging	13
7	Limitations	13
7.1	MPI-over-Grid and clusters	13
7.2	Bandwidth issues	14
7.3	GT2 execution management approach	14
7.4	Required ports	14
7.5	Status of the MPICH-G2 project	14
7.6	Quality of error messages	15

8 Troubleshooting	15
8.1 GRAM job manager log files	15
8.2 Globus 2 job management functionality	15
8.3 Clock skew	16
8.4 Some specific errors	16
9 Open Questions	17
10 Example MPI program	18

1 Introduction to MPI-over-Grid

A Grid user may bring the Grid's vast resources to bear on a single application, by running application processes on multiple compute nodes on the Grid.

Often it is important for these processes to communicate with one another. A very popular standard for such communication is the Message Passing Interface (MPI)[7].

The MPI is routinely used for communication between processes running on nodes of a single cluster, where it utilizes lower-level communication protocols, such as TCP/IP[3].

Note that it is possible using Globus to submit MPI jobs to a cluster, in which case MPI communicates not via Globus, but via the lower-level protocol between the cluster nodes. *This is not* the scenario we discuss in this document.

Here we will discuss how MPI is used to transmit information via Globus between processes running on remote compute resources across the Grid; the compute resources in this scenario serve as MPI nodes.

2 Existing software

In recent years, several community efforts related to MPI over the grid have been initiated. Some include:

- MPICH-G2[9]
This is a special build of the MPICH[8] implementation of MPI. The MPICH-G2 project is listed at the Globus site as "Related Software".
- PACX-MPI[12]
(PARallel Computer eXTension) a library to enable MPI communications over a computational grid.
- IMPI[11]
An effort to create a standard for interoperability of different implementations of the MPI. LAM and GridMPI support IMPI.
- LAM/MPI[10]
Another popular implementation of MPI.
- GridMPI[13]
A new implementation of MPI specifically targeted for computing over an extended "grid" of nodes. An experimental "globusrun Remote Invocation Layer" is listed.

None of the current MPI implementation efforts (particularly, Open MPI and MPICH 2.0) include Globus functionality (although there is sometimes talk of it on mailing lists).

We were unable to obtain the PACX-MPI software while this document was being written, due to problems with the HLRS servers.

The GridMPI effort seems quite active, but the Globus implementation is experimental.

This document will focus on the first implementation, MPICH-G2.

3 Default Procedures for MPICH-G2

For a working MPI there are some requirements to operate: since each of the grid nodes is communicating with another, a sufficient range of ports for this communication has to be available. The range of these ports is set by `GLOBUS_TCP_PORT_RANGE`. For each node to node communication, the MPI processes open up a different port, i.e. the number of required ports is directly related to the number of involved processes.

In D-Grid, there is currently a port range from 20000 to 25000 agreed between the participants, so it would be possible to run a MPI job on roughly 5000 grid nodes if there were this amount of nodes accessible via a public IP number.

Since MPI is intrinsically a parallel communications system, it is sensitive to clock skew. A properly configured NTP (network time protocol) client is recommended on all resources, to keep their clocks synchronized.

For the practical daily use of MPI over the grid, some grid resource manager would be required. This manager would determine which machines were appropriate for a CPU-intensive process, and which had appropriate architectures, software, disk space, etc. for a given job, and also schedule multiple jobs.

The testing was done mostly with the SL (Scientific Linux) 5 operating system[5].

4 Installation of MPICH-G2

MPICH-G2 has to be installed on any machine where executable MPI binaries are to be built.

4.1 Required software

- An installation the Globus Toolkit from source.

We used Globus 4.0.5. Note that Linux distribution packages for Globus will sometimes not contain the required development files needed to build MPICH-G2. Note also that you have to use the Globus source packages.

- A recent release of MPICH

We used MPICH 1.2.7p1.

4.2 Preparation

The MPICH-G2 build mechanism relies on information from the local Globus installation, which should be determined from the start:

- Determine the installed Globus *flavors*

```
ls $GLOBUS_LOCATION/etc/globus_core
```

This produces a list of files like

```
flavor_gcc32dbg.gpt flavor_gcc32dbgpthr.gpt
```

The flavor names are the parts between “flavor_” and “.gpt”.

Note that if the `globus_core` sub-directory is missing, the Globus installation was probably from a pre-built package rather than a source distribution, as required.

- Define the variable `GLOBUS_INSTALL_PATH` to be `$GLOBUS_LOCATION`
- Define the variable `GPT_LOCATION` to be `$GLOBUS_LOCATION`
- Choose an installation directory for MPICH

The default is the current directory, which is not generally desirable. Often an alternative such as `/opt/mpich/1.2.7p1` is used. For some purposes, one can also use the user’s home directory `$HOME`.

We will call the MPICH installation directory `$MPICH_HOME`. It is convenient also to define this variable.

- (IBM AIX/POWER systems only)

On IBM AIX/POWER systems, an additional environment variable, `$OBJECT_MODE`, must be set to 64. Otherwise, one gets an error message

```
Error: mpichg2 cannot be used as the name for the MPICH library
```

4.3 Configure

From within the MPICH build directory, type a line like

```
./configure --prefix=$MPICH_HOME --with-device=globus2:-flavor=gcc64dbg
```

Here, the option `--prefix` specifies the alternative installation directory.

The option `--with-device` turns on the Globus functionality, and the option `-flavor` specifies the chosen Globus flavor.

It is recommended not to specify a threaded flavor unless you know you need it. Flavors with names containing `pthr` are threaded.

4.4 Build and Install

To build the MPICH-G2 libraries and programs, type

```
make
```

To install the software in any system directory, you will probably need to become user `root`. Then type

```
make install
```

Any MPICH users will need their `$PATH` environment variable to contain `$MPICH_HOME/bin`, and their `$LD_LIBRARY_PATH` to contain `$MPICH_HOME/lib`. It is also polite to provide in their `$MANPATH` the directory `$MPICH_HOME/man`

5 Build an MPI test program

A build of an executable program with MPICH-G2, involves libraries and header files from both the MPICH and the Globus installations.

Note that *a binary must be built for each combination of machine architecture and Globus flavor on which the job is to run.*

5.1 Example on a Linux system

For example, to build a simple test MPI program from a C source file `mpitest.c` using the Globus flavor `gcc64dbg`, a command like this would be used:

```
gcc -o test mpitest.c -I $MPICH_HOME/include -L $MPICH_HOME/lib \
-L $GLOBUS_LOCATION/lib -lmpichg2 -lglobus_common_gcc64dbg \
-lglobus_ftp_client_gcc64dbg -lglobus_nexus_gcc64dbg \
-lglobus_duroc_runtime_gcc64dbg
```

This example will build an executable program named `test`.

Note that the names of the Globus libraries reflect the chosen flavor of Globus. (Here, the ending `_gcc64dbg` reflects the flavor.)

5.2 Example on an IBM AIX/POWER system

The build line using the IBM `xlc` compiler needs to link in many more libraries:

```
xlc -o hello mpihello.c \
-I $MPICH_HOME/include -L $MPICH_HOME/lib -lmpichg2 \
-L $GLOBUS_LOCATION/lib \
-lglobus_ftp_client_vendorcc64 -lglobus_ftp_control_vendorcc64 \
-lglobus_io_vendorcc64 -lglobus_xio_vendorcc64 \
-lglobus_duroc_runtime_vendorcc64 -lglobus_duroc_common_vendorcc64 \
-lglobus_dc_vendorcc64 -lglobus_nexus_vendorcc64 \
-lglobus_gram_client_vendorcc64 -lglobus_gram_protocol_vendorcc64 \
-lglobus_gram_myjob_vendorcc64 \
-lglobus_gssapi_gsi_vendorcc64 -lglobus_gsi_proxy_core_vendorcc64 \
-lglobus_gsi_cert_utils_vendorcc64 -lglobus_gsi_credential_vendorcc64 \
-lglobus_gsi_callback_vendorcc64 -lglobus_gss_assist_vendorcc64 \
-lglobus_gsi_sysconfig_vendorcc64 -lgssapi_error_vendorcc64 \
-lglobus_rsl_vendorcc64 -lssl_vendorcc64 -lcrypto_vendorcc64 \
-lglobus_proxy_ssl_vendorcc64 \
-lglobus_openssl_vendorcc64 -lglobus_openssl_error_vendorcc64 \
-lglobus_duct_runtime_vendorcc64 \
-lglobus_callout_vendorcc64 -lglobus_oldgaa_vendorcc64 \
-lglobus_common_vendorcc64 \
-lltdl_vendorcc64 -lm
```

If the link still fails with undefined symbols, you have to search for the libraries where they are defined. For those that come from Globus, use `grep` to find the symbols in the `lib*.a` files in `$GLOBUS_LOCATION/lib`, then with

```
nm library_name.a | less
```

search for the symbol in each library found. If the symbol is marked with a `U`, it is undefined in that file. Find a file where the symbol is marked with a `T` or a `D`. This file can then be added to the link line.

5.3 Example with `mpicc`

When installed, `mpich` generates a set of scripts, which know the right compiler-switches and libraries. If the path to `mpichG2` is exported as described above, compiling the testprogram could be done on any architecture with

```
mpicc -o test -c test.c
```

which should normally do all the linking correctly, producing the program `test`. There are makefile examples in the source package, giving directions for Fortran programs as well.

6 Run an MPI program over the Grid

To run an MPI program, one needs first to produce a list of grid compute resources on which instances of the executable are to be run.

In the simplest case, the compute resources must all have architectures compatible with, and the same Globus flavor as, the executable file to be run. However, note that it is *not* necessary that MPICH be installed on the compute resources (the MPICH code is statically linked in the executable binaries).

A program called `mpirun` comes with MPICH. It is used to start the multiple instances of the program running on the various resources. This program can be called from any grid-enabled machine with MPICH-G2 installed, and where the user has a validated their GSI credential (by running `grid-proxy-init`).

There are several ways to arrange for the program to run, but they all achieve the same sequence of events.

- The executable, data and any supporting files are copied to each node.
- `mpirun` logs in to each node, and starts the requested number of instances of the executable on each node.
- `mpirun` creates an *MPI world*, by which running processes refer to one another.
- processes exchange data with other processes, by making MPI communications function calls.

- the MPI communications function calls transmit the data across the grid via Globus communications facilities.

To run a simple code, which has been compiled and linked with the MPI and Globus libraries, the user does the following:

- makes a `machines` file containing a list of desired compute resources, in the format described below
- copies the executable binary and supporting files to each of the resources
- calls `mpirun` with the desired number of processes and the executable name as arguments

6.1 The machines file

The `machines` file is plain text with two columns. The first contains the resource URL, and the second contains the maximum number of processes to start on that resource (the default being 1).

The number in the second column of the `machines` file usually reflects the number of computing cores on that resource. For example, if the resource were a workstation with a dual-core CPU, one would put "2" in the line corresponding to that resource. If the resource were a PC farm of 10 nodes each of which has two dual-core CPUs, one would put "40" for this number.

6.2 Globus modules involved

In what follows, several Globus modules[6] become involved. It is useful at least to know their basic functions, as the acronyms come up often in the documentation, in discussions of RSL files, and in the log files.

- GRAM *Grid Resource Allocation and Management*
submits a single job for execution to a specified job manager.
- DUROC *Dynamically-Updated Request Online Coallocator*
submits several simultaneous jobs to different job managers
- GASS *Global Access to Secondary Storage*
stages files from a remote resource (by default, stages `stdout` and `stderr`.)

6.3 Runs on heterogeneous grids with RSL scripts

The direct use of the `mpirun` command is only useful in a grid where all the systems on the nodes are identical. If any file path on any node is different from another, it will fail. But MPICH-G2 makes provisions for more heterogeneous grids.

MPICH-G2 communicates its job management requirements via the Globus RSL (*Resource Specification Language*). Job management can be done in a much more flexible way by editing an RSL script and running MPICH-G2 on that script.

To see the RSL script MPICH-G2 produces by default for given input, run the `mpirun` command with the `-dumprsl` option, for example,

```
$ mpirun -dumprsl -np 4 mpitest/hello > myrun.rsl
```

will produce an RSL script in `myrun.rsl` that might look something like this:

```
+
( &(resourceManagerContact="gavo2")
  (count=2)
  (label="subjob 0")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
    (LD_LIBRARY_PATH /usr/local/globus/gtk/lib/))
  (directory="/work1/gridspace/agdusr039/mpitest")
  (executable="/work1/gridspace/agdusr039/mpitest/hello")
)
( &(resourceManagerContact="astrodata01")
  (count=2)
  (label="subjob 2")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
    (LD_LIBRARY_PATH /usr/local/globus/gtk/lib/))
  (directory="/work1/gridspace/agdusr039/mpitest")
  (executable="/work1/gridspace/agdusr039/mpitest/hello")
)
```

The default paths in the `myrun.rsl` file all reflect paths to libraries and executables on the system where the command was run. These may be altered to be appropriate for the target systems. Also, environment variables for the target system may easily be added.

Note that the script will run on the target resources in a restricted environment, and it may take some effort to determine which environment variables are required. In particular, care must be taken to ensure that the `PATH` and `LD_LIBRARY_PATH` variables refer to the correct directories for Globus on each system. Also, the `GLOBUS_TCP_PORT_RANGE` variable must be set to `20000,25000` on each system.

To determine just what this restricted environment contains, the executable in the RSL script may be replaced with a shell script that writes the current environment to a file.

Here is a working example of an RSL script modified to run an MPICH-G2 job on machines at two institutes (call it `hello.rsl`):

```
+
( &(resourceManagerContact="alnitak.ari.uni-heidelberg.de")
  (count=3)
  (label="subjob 0")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
    (GLOBUS_HOSTNAME alnitak.ari.uni-heidelberg.de)
    (GLOBUS_TCP_PORT_RANGE 20000,25000)
    (LD_LIBRARY_PATH /opt/d-grid/globus/gt405/lib/))
)
```

```
(directory="/home/Agrid/agrid039/mpitest")
(executable="/home/Agrid/agrid039/mpitest/hello")
)
( &(resourceManagerContact="astrodata01.gac-grid.org")
(count=2)
(label="subjob 3")
(environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
(GLOBUS_HOSTNAME astrodata01.gac-grid.org)
(GLOBUS_TCP_PORT_RANGE 20000,25000)
(LD_LIBRARY_PATH /usr/local/globus/gtk/lib/))
(directory="/work1/gridspace/agdusr039/mpitest")
(executable="/work1/gridspace/agdusr039/mpitest/hello")
)
```

The job is run from this script simply by

```
$ mpirun -globusrsl hello.rsl
```

6.4 File staging

Judging from the existing documentation, it should be possible to use GASS to copy the executable file from the initiating account to the target resources, using an executable line in the RSL script like

```
(executable="$(GLOBUSRUN_GASS_URL)/full_path/program")
```

However, many experiments with this procedure have failed.

One can of course write a script to stage the executables to the appropriate resources, to be run before the `mpirun` command.

7 Limitations

7.1 MPI-over-Grid and clusters

MPI processes communicate during execution, and, an MPI-over-Grid job is typically running on multiple resources on the grid. Unless all the processes on all the resources are running at the same time, they cannot communicate.

However, a cluster resource manager will naturally delay starting of job processes on the cluster. If those processes are part of an MPI-over-Grid job, they can't communicate with sister processes running on some other grid resource.

Therefore, MPI-over-Grid and cluster MPI jobs are to be considered as incompatible alternatives to one another.

7.2 Bandwidth issues

Some traditional uses of MPI in clusters are inappropriate for use on an extended grid. While most clusters have fast, dedicated network interconnects between their nodes, providing very high bandwidth for communication between processes in MPI jobs, a typical grid will have relatively slower communication.

This means, applications making good use of MPI-over-Grid will be those that require much less inter-process bandwidth than applications that require a cluster.

Also, MPICH-G2 has some mechanisms for taking advantage of grid topology (that is, making use of knowledge of the relative speeds of node interconnects to optimize use of bandwidth.) The usefulness of such a mechanism depends on the application, however.

7.3 GT2 execution management approach

The current MPICH-G2 uses the Globus 2.x Resource Allocation Manager (GRAM) approach [6] to job execution management, as opposed to the Globus 4.x Web Services (WS) approach.

This is limiting, because for example accounting mechanisms that rely on Globus WS will not account for MPICH-G2 jobs.

Furthermore, the administrators of some grid resources have disabled the older approach, in order to enforce the running of grid jobs only through a cluster job batch system, as opposed to forking jobs directly. Such grid resources cannot be used as compute nodes for MPICH-G2.

There has been some talk of a new release of MPICH-G2 that would support Globus WS, but as of this writing, no such release has appeared.

7.4 Required ports

The current MPICH-G2 version requires that the Globus ports (in D-Grid, by convention, ports 20000 through 25000) should be open for communications between all systems involved.

Once again, this limitation is due to the use of the GT 2.0 execution management model, and would go away if a Globus Web Services model were employed.

7.5 Status of the MPICH-G2 project

As of this writing, no changes have been made to the MPICH-G2 web site since December 2005, and it is badly out of date with many broken links.

The Globus Bugzilla lists no bugs fixed in the MPICH-G2 project since September 2004.

The mailing list `mpich-g@globus.org` still had occasional activity until the third quarter of 2007, and has been mostly dormant since. No useful responses to several queries made by us have been forthcoming. Some support for use of the software would be very helpful.

7.6 Quality of error messages

One of the greatest difficulties in getting started with MPICH-G2 is that many things can go wrong without presenting any error messages. One will just see the job fail to run.

If the job is submitted, but fails to run, the GRAM job manager will drop a log file, but the error messages there are very difficult to interpret.

This can be seen as a weakness in the Globus 2 job management system. Perhaps this too would be improved if MPICH-G2 were upgraded to use Globus Web Services job management.

8 Troubleshooting

8.1 GRAM job manager log files

Each job that is submitted but somehow fails to run will result in a GRAM job manager log file in the user directory on the target machine. These are named

```
gram_job_mgr_XXXXX.log
```

They are a little hard to read, but in particular contain the RSL script as received by the job manager.

Several kinds of errors in job submission can be read out of these files, especially, failure to find a directory or executable, and failure to open stdout.

8.2 Globus 2 job management functionality

To tell if the Pre-WS GRAM functionality is available on a resource, run on that resource

```
globusrun -a -r machine-url
```

It should report that the GRAM Authentication test was successful.

An independent check of the Globus 2 job management system can be made so:

```
globusrun -o -r TARGET_MACHINE '&(executable=/bin/date)'
```

If it times out with an error

```
GRAM Job submission failed because the job manager failed to open stderr  
(error code 74)
```

Then look at the resulting GRAM log files on the target machine, and check that the ports are within the specified GLOBUS_TCP_PORT_RANGE (check the value of the port in GLOBUSRUN_GASS_URL. A wrongly-formatted string for the port range, such as 20000-25000 instead of 20000,25000, can cause such a failure.

It may also be that the value returned by the `globus-hostname` command is not a fully-qualified domain name. This may be fixed by setting the GLOBUS_HOSTNAME environment variable.

8.3 Clock skew

MPI communications are sensitive to clock skew.

A properly configured NTP (network time protocol) client is recommended on all resources, to keep their clocks synchronized.

8.4 Some specific errors

- Fails with no output on initiating machine if executable isn't present

For a simple non-staging job, where executable hasn't been copied to target machines, or is in the wrong place `mpirun` just exits with no output.

In the `gram_job_mgr` log file on the target machine, will see

```
failure-code: 94
```

- When IP address is wrong, or ports are closed, get this error:

```
Submission of subjob (label = "subjob 1") failed because the connection to the server failed (check host and port) (error code 62)
```

For example, the target machine has not opened the required ports, and isn't running the Globus 2.x Gatekeeper. Or, simply specifying the wrong IP address in the machines file will produce this result.

- An otherwise successful submission that fails because it couldn't establish communication through a TCP port may give an error like this:

```
Submission of subjob (label = "subjob 1") failed because the job manager failed to open stderr (error code 124)
```

- GRAM Authentication test failure: the connection to the server failed (check host and port)

This could happen if the address of the machine is wrong. We have also seen it when the machine isn't running a Globus 2 gatekeeper.

- Globus TCP port range not properly set.

Job submission eventually times out with an error

```
GRAM Job submission failed because the job manager failed to open stderr (error code 74)
```

The `gram_job_mgr` log files on targets had a line containing an indication that the port used was out of range, for example:

```
Pre-parsed RSL string: &("rsl_substitution" = ("GLOBUSRUN_GASS_URL" "https://host-address:54615" ) )
```

Since the jobs run in a restricted environment, the range for Globus TCP ports should be explicitly set for each node in the RSL script's environment variable setting.

```
(GLOBUS_TCP_PORT_RANGE 20000,25000)
```

Note: the wrong syntax for the port range (e.g.: 20000-25000) causes the same problem.

9 Open Questions

- Due to resource limitations, no tests of transmission of data over the grid by MPI were performed. Such tests have been done before, when MPICH-G2 was being developed, but since this is the essential functionality of MPI, tests should be repeated with modern Globus on modern systems.
- One error message, saying job submission failed with error code 12, occurred on a couple of resources and no others. It would be good to investigate that.
- It would also be good to investigate the symptoms of clock skew.

10 Example MPI program

The following is the C code for a very rudimentary MPI program, that does nothing but initialize the MPI “world” and print out the basic parameters: the number of processes (the “size”) and the MPI process number (the “rank”).

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

int
main( int argc, char **argv )
{
    int rank, size;
    char hostname[HOST_NAME_MAX] = { '\0' };

    gethostname( hostname, sizeof( hostname ) ) ;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );

    fprintf( stdout, "MPI world of size %d inited on %s, rank %d.\n",
             size, hostname, rank );

    MPI_Finalize();
    return 0;
}
```

Below is sample output of the code, run on the machine *astrodata* which has numbered nodes each of which with two processors, using the command

```
mpirun -np 4 hello
```

(Note that the hello binary must already have been copied to the execution nodes.)

```
MPI world of size 4 inited on astrodata01.gac-grid.org, rank 0.
MPI world of size 4 inited on astrodata01.gac-grid.org, rank 1.
MPI world of size 4 inited on astrodata02.gac-grid.org, rank 2.
MPI world of size 4 inited on astrodata02.gac-grid.org, rank 3.
```

Here, the output lines happen to be ordered by MPI rank, but that will not typically be the case.

References

- [1] Deliverable 1.2 (AGD-WG1), *Integration of Astrogrid-D Resources*, http://www.gac-grid.org/project-documents/deliverables/wp1/ResourceIntegration-D_1_2_1.0.0.pdf
- [2] Deliverable 1.3 (AGD-WG1), *Resource Integration and Grid Support* http://www.gac-grid.org/project-documents/deliverables/wp1/Working_AGD_Grid-D_1_3_1.0.1.pdf
- [3] Deliverable 1.4 (AGD-WG1), *Running MPI Jobs on Grid Resources* (under construction)
- [4] Deliverable 5.1 (AGD-WG5) *Ressourcen-Management für Grid-Jobs*, <http://www.gac-grid.org/project-documents/deliverables/wp5/D5.1.pdf>
- [5] Scientific Linux <https://www.scientificlinux.org/>
- [6] Globus Toolkit 2.4 Manuals <http://www.globus.org/toolkit/docs/2.4/>
- [7] MPI Standard <http://www-unix.mcs.anl.gov/mpi/>
- [8] MPICH <http://www-unix.mcs.anl.gov/mpi/mpich1/>
- [9] MPICH-G2 <http://www3.niu.edu/mpi/>
- [10] LAM/MPI <http://www.lam-mpi.org/>
- [11] IMPI <http://impi.nist.gov/>
- [12] PACX/MPI <http://www.hlr.de/organization/amt/projects/pacx-mpi/>
- [13] GridMPI <http://www.gridmpi.org/>