



CONTRACT NUMBER 508830

**DEISA**  
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR  
SUPERCOMPUTING APPLICATIONS**

**European Community Sixth Framework Programme**  
**RESEARCH INFRASTRUCTURES**  
Integrated Infrastructure Initiative

Documentation for D-JRA2-1.2

Deliverable ID: D-JRA2-1.3  
Due date: October, 31, 2005  
Actual delivery date: November 28, 2005  
Lead contractor for this deliverable: EPCC, UK

Project start date: May 1<sup>st</sup>, 2004  
Duration: 4 years

<b>Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)</b>		
<b>Dissemination Level</b>		
<b>PU</b>	Public	X
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	

## Document Keywords and Abstract

<b>Keywords:</b>	Virgo, Cosmology, GADGET, Grid, Message Passing Interface MPI, PACX-MPI, MPICH-G2, meta-computing
<b>Abstract:</b>	<p>The steps required to run the GADGET code using MPI in a meta-computer are described. These steps include installation and compilation of the PACX MPI and MPICH-G2 libraries, and a number of special measures required for the DEISA infrastructure.</p> <p>The additions required to the publicly available GADGET2 code to run using PACX MPI or MPICH-G2 are described. Compilation and run time issues are also discussed. The DEISA Grid-aware decomposition code is available from the author.</p>

---

**Table of Content**

<b>1 INTRODUCTION .....</b>	<b>2</b>
1.1 Executive Summary .....	2
1.2 References and Applicable Documents.....	2
1.3 List of Acronyms and Abbreviations .....	2
<b>2 GRID MPI IMPLEMENTATIONS .....</b>	<b>3</b>
2.1 PACX MPI.....	3
2.1.1 Installation .....	3
2.1.2 Compilation against PACX MPI.....	3
2.1.3 General Usage .....	3
2.1.4 Files.....	4
2.1.5 Run Time Environment.....	5
2.1.6 DEISA Infrastructure.....	5
2.2 MPICH-G2 .....	5
2.2.1 Installation .....	5
2.2.2 Compilation Against MPICH-G2.....	6
2.2.3 General Usage .....	6
2.2.4 Run Time Environment.....	6
2.2.5 DEISA Infrastructure.....	6
<b>3 DEISA DECOMPOSITION CODE IN GADGET .....</b>	<b>6</b>
3.1 Prerequisites.....	6
3.1.1 GSL 1.5 .....	6
3.1.2 FFTW 2.1.5.....	7
3.2 Initialisation .....	7
3.3 The Decomposition Stage .....	7
3.4 Comments on Usage .....	7
3.4.1 Preprocessor Options.....	7
3.4.2 Compiler Options.....	7
3.4.3 DEISA Infrastructure.....	8
<b>4 CONCLUSION.....</b>	<b>8</b>

## 1 Introduction

### 1.1 Executive Summary

This document is aimed at both users and administrators, who must work together, to run the grid-enabled version of GADGET2 on the DEISA infrastructure.

This document describes the practical steps required to run the GADGET2 code using either PACX MPI or MPICH-G2, both “Grid-enabled” Message Passing Interface libraries. Some of the practical difficulties in each case are discussed.

In practice, MPICH-G2 would be the preferable solution for running Grid-enabled MPI tasks, along with appropriate co-scheduling of computational resources. It is slightly more flexible to use than PACX MPI, and may offer better performance.

The additions required to allow GADGET to run in a Grid environment are described. The DEISA decomposition code, based on the publicly available version of GADGET2, is available from the author.

As described in [1], the resultant performance is poor and it is recommended not run this version of GADGET in a metacomputing environment until the issues given here and in [1] have been addressed.

### 1.2 References and Applicable Documents

- [1] D-JRA2-1.2 Grid-enabled implementation of GADGET for metacomputing environments
- [2] Fastest Fourier Transform in the West <http://www.fftw.org/>.
- [3] GADGET <http://www.mpa-garching.mpg.de/gadget/>.
- [4] Globus Toolkit <http://www.globus.org/>.
- [5] Gnu Scientific Library <http://www.gnu.org/software/gsl/>.
- [6] Karonis, N.T., Toonen, B., and Foster, I., MPICH-G2: A Grid-enabled implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing*, **63**, 551-563, 2003.
- [7] PACX MPI <http://www.hlrs.de/organization/pds/projects/pacx-mpi/>.

### 1.3 List of Acronyms and Abbreviations

<b>GADGET</b>	<b>GA</b> laxies with <b>D</b> ark matter and <b>G</b> as intEracT
<b>MPI</b>	<b>M</b> essage <b>P</b> assing <b>I</b> nterface
<b>MPICH-G2</b>	<b>M</b> PI <b>CH</b> ameleon <b>G</b> lobus 2
<b>PACX MPI</b>	<b>PA</b> rallel <b>C</b> omputer <b>eX</b> tension <b>M</b> PI

## 2 Grid MPI Implementations

Two Grid-enabled MPI implementations are discussed: PACX MPI [7] and MPICH-G2 [6].

### 2.1 PACX MPI

PACX MPI (version 4.1.4) has been employed for the work undertaken in this report. Investigation of PACX version 5 (PACX 5.0.0rc1, the publicly available version at the time of writing) suggested it was not in a usable state.

#### 2.1.1 Installation

The configuration, compilation and installation of PACX MPI (version 4.1.4) is straightforward, and the instructions provided are adequate. Care is required if one wants to change the default port on which the `startupserver` listens for connections (the relevant include file must be updated before compilation). On IBM AIX systems the Makefile generated during the configure step must be modified to include the correct location of the Fortran include file `mpif.h`, which is in a non-standard location. It is also useful to use the GNU version of `yacc` (`bison`), along with `flex` instead of the default `lex`. PACX MPI can be configured with the `-enable-conversion` option to allow messages to be passed between heterogeneous architectures.

**Important.** On the DEISA infrastructure (IBM AIX) there appeared to be a serious problem associated with the `-enable-compression` option to PACX MPI, which allows large messages to be sent in compressed form. This is interpreted as a bug in PACX MPI 4.1.4 and means that `-enable-compression` should not be used.

#### 2.1.2 Compilation Against PACX MPI

Compilation of an MPI code against PACX MPI can be performed with the usual compiler (`xlc_r`) with appropriate options to ensure the PACX include files are picked up before the standard MPI include files, and that the PACX library (`libpacx.a`) is picked up before the standard MPI libraries. Alternatively, one can use the PACX compiler wrapper `pacxcc`.

On IBM AIX systems, the additional option `-binitfini:poem_remote_main` should be included at compile (strictly, link) time for executables. If this is not done, the executable will not find the correct initialisation routine at run time and may fail with an error of the form

```
INTERNAL ERROR: catalogue not closed..
```

#### 2.1.3 General Usage

Having installed and compiled the same executable on two systems (whether sharing a common file system or not) PACX MPI offers two basic ways of bootstrapping communication between the two sets of MPI processes. While neither is ideal, only one provides a mechanism by which two MPI jobs can be started when the name of the execution host is not known in advance (as is typically the case in a large cluster batch system). This is the `startupserver`, the use of which will be described in this section.

Successful communication clearly relies on there being appropriate network connectivity between the two machines running PACX MPI jobs and requires appropriate port authorisation if there is a firewall between the two. We assume such issues have been addressed.

It can be difficult to diagnose problems when first starting to use PACX MPI. Enabling debugging information via the `PACX_DEBUG_NODE` flag and verbose messages from the native MPI or parallel environment can help. In addition, it is useful to have something (e.g., a simple `perl` script) which allows you to check port connections between the machines involved independently of PACX MPI.

As PACX MPI uses fixed ports for communication, it is usually possible to run only one PACX MPI job at any given time.

#### 2.1.4 Files

PACX MPI uses two files on each machine in which the user provides details of the meta-computing environment. These are by default:

- `~/.hostfile` in the user's home directory, and
- `./netfile` in the current working directory.

In practice, it is preferable to make these files visible. Different files can be used, the location of which is defined by appropriate values for the environment variables `PACX_HOSTFILE` and `PACX_NETFILE`.

The `hostfile` provides the location of the startupserver, so that the two PACX MPI jobs can "rendez-vous" at runtime. An example of the `hostfile` is:

```
Server e3500.epcc.ed.ac.uk 0
```

The final number in the line indicates the order of the PACX MPI job in the meta-computer. This number must be different at the two sites. If not, error messages of the form

```
ERROR on local node: 0; Reason:PACX_netfile_yacc.y:
tmp_host_number: 2 not possible! (PACX_numhosts:1)
```

```
ERROR on local node: 0; Reason:set_comp: couldn't determine
this host in the configuration
```

may occur on the respective machines.

The `netfile` contains details of the network ports and protocol to be used to exchange messages. An example of the `netfile` is:

```
BEGIN 1;
  HOST=2, PROTOCOL=tcp, PORT=65000;
END;
BEGIN 2;
  HOST=1, PROTOCOL=tcp, PORT=65000;
```

```
END;
```

The file is identical on both machines.

### 2.1.5 Run Time Environment

In addition to `PACX_HOSTFILE` and `PACX_NETFILE`, which determine the whereabouts of information about the meta-computer, the environment variable `LOCALDOMAIN` should be set appropriately at each site to ensure PACX MPI constructs the correct network addresses at run time. For example, at IDRIS

```
LOCALDOMAIN=idris.fr
```

while at RZG

```
LOCALDOMAIN=rzg.mpg.de
```

For IBM AIX machines, one should also ensure that MPI messages are not transferred via shared memory, i.e., ensure that `MP_SHARED_MEMORY=no`, as there is no support in PACX MPI for shared memory. If this is omitted, errors of the form

```
ERROR: 0032-171 Communication subsystem error: Destination  
task is purged. In MPI_Bsend, task 2
```

may occur.

### 2.1.6 DEISA Infrastructure

In order to run PACX MPI jobs over the DEISA network, the correct network addresses for the machines involved must be available at run time. At IDRIS, the correct network address for machines `zahirNNN.idris.fr` is constructed by appending `-dml0` to the hostname, e.g., for `zahir001.idris.fr` the appropriate DEISA network address would be `zahir001-dml0.idris.fr`.

The same mapping at RZG is to append `s` to the hostname, e.g., for the machine `psi24.rzg.mpg.de` the appropriate network address is `psi24s.rzg.mpg.de`.

These mappings were performed by adding a line to the PACX MPI source to append the appropriate letter or string to the hostname as returned by `gethostname()`.

As no co-scheduling was possible, batch jobs to run PACX MPI tests between the two sites must be started simultaneously by special arrangement between the operational staff at each site.

## 2.2 MPICH-G2

The installation of MPICH-G2 was undertaken using Globus version 2.4.3, and MPICH-G2 version 1.2.7.

### 2.2.1 Installation

The installation procedure for MPICH-G2 is considerably more complex than that for PACX MPI. The essential prerequisite is the Globus Toolkit [4], which itself provides a considerable overhead in effort. If the native MPI is to be available for intra-machine

communication, then Globus must be built with the appropriate MPI compilers and libraries. For IBM AIX systems this proved difficult as several patches for the Globus 2.4.3 source were required (some of which were obtained from more recent releases of Globus and 'back-ported' to 2.4.3). It would be hoped that building any newer version would be easier.

A number of GNU packages (`make`, `sed`, `tar`, along with special versions of the GNU autotools) were also required in order to build the Globus Toolkit. Special effort was required on AIX into re-writing parts of the interface between the Globus Job Manager and LoadLeveler to ensure that jobs submitted via Globus were able to run successfully.

Having built the Globus Toolkit, the MPICH-G2 libraries can be compiled in a reasonably straightforward manner.

### 2.2.2 *Compilation Against MPICH-G2*

Compilation of MPI applications against the MPICH-G2 library can be carried out using the `mpicc` compiler wrapper provided. Alternatively, one can arrange that the appropriate include files and libraries are picked up before the native files manually.

### 2.2.3 *General Usage*

MPICH-G2 requires a valid Globus-compatible X.509 digital certificate to be held by the user.

### 2.2.4 *Run Time Environment*

Details of the job to be run are specified in a Globus Resource Specification Language (RSL) file. This specifies the number of processors, location of executable, and so on that are to be used on the different machines. The RSL file is submitted using the single command

```
globusrun -f <file.rsl>
```

Appropriate batch submission is handled by Globus. A number of MPICH-G2 jobs can be run at any one time, as Globus uses a range of ports at run time. Globus mechanisms can also be used to manage and collect standard output and error from each of the machines.

### 2.2.5 *DEISA Infrastructure*

It is not envisaged that MPICH-G2 will be available on the DEISA infrastructure in the near future.

## **3 DEISA Decomposition Code in GADGET**

The publicly available version of GADGET2 [3] has been adapted to use with PACX MPI and MPICH-G2. The new code is called the DEISA Decomposition Code.

### **3.1 Prerequisites**

#### 3.1.1 *GSL 1.5*

The Gnu Scientific Library (GSL, [2]) is required by GADGET. This is a purely serial library, and can be compiled in the normal way with appropriate compilation options. Version 1.5 has been used throughout this work.

### 3.1.2 FFTW 2.1.5

The Fastest Fourier Transform in the West (FFTW, [2]) library is required in its parallel implementation (configure with `--enable-mpi`) by GADGET. It must be ensured that FFTW is built against the correct MPI library (PACX MPI or MPICH-G2), and again with consistent compiler options. Version 2.1.5 is used throughout.

## 3.2 Initialisation

Initialisation of the DEISA decomposition code is performed by the routine `deisa_init()`, which is called from `main.c` after `MPI_Init()`. This initialises the information required about the meta-computer depending on the MPI implementation selected at compile time.

## 3.3 The Decomposition Stage

The DEISA decomposition replaces calls to `domain_findSplit()` and `domain_shiftSplit()` and some of the surrounding code in `domain_decompose()` by a call to the new routine `deisa_clusterSplit()`. This routine has exactly the same result as the original code in that it computes the new decomposition and sets the values of the global tables `DomainTask[]`, `DomainStartTask[]`, and `DomainEndTask[]`, along with the local variables `MyDomainStart` and `MyDomainLast`. There are no other side-effects.

Currently, the DEISA decomposition will identify up to two different clusters and perform the decomposition appropriately. There was no reason to generalise the code to support a larger meta-computer at the current time.

## 3.4 Comments on Usage

### 3.4.1 Preprocessor Options

The DEISA decomposition code can be introduced by specifying the pre-processor directive `-D_DEISA_` at compile time. In addition, the MPI implementation must be specified as one of

```
-D_DEISA_PACX_MPI_
```

or

```
-D_DEISA_MPICH_G2_
```

If neither of these options is specified, the DEISA code will assume native MPI will be used and the number of clusters will default to one. This can be used if one wants to employ the DEISA decomposition code in normal usage.

### 3.4.2 Compiler Options

It should be ensured that GADGET picks up the correct MPI library, which is most easily ensured using the compiler wrappers supplied with the Grid-enabled MPI implementations. The code must also be linked against the version of FFTW compiled in like fashion, i.e., with the appropriate MPI.

For IBM AIX, the following options were used:

```
CC      = xlc_r -binitfini:poe_remote_main -qlanglvl=extended \  
-q64 -qipa
```

```
OPTIMIZE = -O5 -qhot -qarch=pwr4 -qtune=pwr4 -qstrict
```

### 3.4.3 DEISA Infrastructure

The default compiler (provided via `module load deisa`) at the time of writing was IBM C version 7.0.

## 4 Conclusion

This document has described the steps required to be run by administrators and users, in order to run GADGET on the DEISA infrastructure using a metacomputing paradigm.

However, as described in [1], the resultant performance is poor and it is recommended not run this version of GADGET in a metacomputing environment until the issues given here and in [1] have been addressed.