

CONTRACT NUMBER 508830

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

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

Specifications document for D-JRA2-1.2

Deliverable ID: D-JRA2-1.1  
Due date: December, 31, 2004  
Actual delivery date: January, 6, 2005  
Lead contractor for this deliverable: EPCC, UK

Project start date: May 1<sup>st</sup>, 2004  
Duration: 5 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 Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	

---

## Table of Content

Table of Content .....	1
1. Introduction .....	2
1.1 Executive Summary .....	2
1.2 References and Applicable Documents .....	2
1.3 List of Acronyms and Abbreviations .....	4
2. Grid MPI.....	5
2.1 Overview.....	5
2.1.1 Practical issues.....	6
2.1.2 Document overview .....	6
2.2 Standards .....	7
2.2.1 Interoperable MPI.....	7
2.3 Grid-Enabled MPI Implementations .....	7
2.3.1 MPICH-G2.....	7
2.3.1.1 Related projects .....	9
2.3.2 LAM/MPI.....	9
2.3.3 PACX-MPI.....	9
2.3.4 StaMPI.....	10
2.3.5 Related issues.....	10
2.3.5.1 Open MPI.....	10
2.3.5.2 MagPle.....	10
2.3.5.3 PVM.....	11
2.4 Summary .....	11
3. GADGET .....	11
3.1 The GADGET Code .....	11
3.2 Compilation and Testing .....	12
3.3 Performance on HPCx.....	12
3.3.1 The qsort routine .....	12
3.3.2 MPI Performance.....	13
4. Grid-enabling GADGET .....	14
4.1 Meta-computing.....	14
4.1.1 Decoupling SPH from the gravitational computation. ....	14
4.1.2 Decoupling the Treecode and the PM sections. ....	15
4.1.3 Decoupling the sub-trees of the Treecode. ....	15
4.2 Single Site Optimisations .....	15
4.2.1 Self Tuning.....	15
4.2.2 Optimising Communications.....	15
4.2.2.1 Collective Communications .....	15
4.2.2.2 Blocking versus Non-Blocking Communications.....	15
4.2.2.3 Removing Barriers .....	16
4.3 Homogeneous and Zoom Simulations .....	16
5. Conclusion and Future Work .....	16
5.1 Comparing PACX-MPI and MPICH-G2 .....	16
5.2 Porting GADGET .....	16
5.3 Tuning GADGET .....	16
5.4 Zoom simulations .....	17
5.5 Optimising Communication Routines.....	17
5.6 Introducing Self-Tuning.....	17

# 1. Introduction

## 1.1 Executive Summary

The aim of this work package, WP1 of JRA2 is to port and Grid-enable the cosmological simulation code GADGET 2.0 to allow the code to run over the DEISA infrastructure. This 'specification' document describes the work done and the work remaining required by WP1. As such, this is not a *technical* specifications document.

Our intention is to run GADGET in a meta-computing manner, using MPI to communicate between different supercomputing sites within the DEISA infrastructure. An overview of currently available Grid MPI implementations is provided that will allow GADGET 2.0, already an MPI code, to fully exploit the DEISA Grid infrastructure. As such, this report recommends that at least MPICH-G2 and PACX-MPI are made available on the DEISA infrastructure to allow a comparison of both these Grid MPI implementations, to determine best suited for running GADGET 2.0 over the DEISA infrastructure.

The testing and optimisation of GADGET2, which has been performed on an IBM p690+ cluster, are described.

A description of future work for the Grid-enabled version of GADGET 2.0 is provided. Once GADGET has been ported, and assuming proper provision of the underlying infrastructure, an evaluation of how effectively GADGET 2.0 runs on the DEISA infrastructure will be made. Time will be spent tuning the code, portioning different parts of the computation to different sites, to investigate the best method of distributing work. This will involve introducing new code to GADGET 2.0

Once the code is running satisfactorily, some further code alterations will be considered, including refactoring communication code segments, introducing 'self-tuning' and determining what type of cosmological simulations best suite GADGET 2.0 within the DEISA infrastructure.

We thank Volker Springel and Denis Girou for ideas and comments.

## 1.2 References and Applicable Documents

- [1] Barnes, J., and P. Hut, *Nature*, **324**, 1986.
- [2] Burns, G., Daoud, R. and Vaigl, J., LAM: An Open Cluster Environment for MPI, *Proceedings of Supercomputing Symposium*, 379--386, 1994. Available from <http://www.lam-mpi.org/papers/>.
- [3] <http://ccse.koma.jaeri.go.jp/>.
- [4] Chin, J., and Coveney, P. V., Towards tractable toolkits for the Grid: a plea for lightweight, usable middleware, UK e-Science Technical Report, number UKeS-2004-01. Available from <http://www.realitygrid.org/>.
- [5] <http://www.cs.vu.nl/albatross/>.
- [6] <http://www.etnus.com/>.
- [7] <http://www.fft.w.org/>.

- 
- [8] Foster, I., and Kesselman, C., Eds. "The Grid: Blueprint for a New Computing Infrastructure" Morgan Kaufman Publishers, 1999.
  - [9] George, W.L., Hagedorn, J.G. and Devaney, J.E., IMPI: Making MPI Interoperable, *J. Res. Natl. Inst. Stand. Technol.*, **105**, 343–348, 2000. Available from <http://impi.nist.gov/>.
  - [10] <http://www.globus.org/>.
  - [11] <http://www.gnu.org/software/gsl/>.
  - [12] Hein J., Booth, S., Bull, M., Exchanging multiple messages via MPI, HPCx technical report, HPCxTR0308, 2003.
  - [13] <http://www.hpcx.ac.uk>
  - [14] Imamura, T., Tsujita, Y., Koide, H. and Takemiya, H., An architecture of Stampi: MPI library on a cluster of parallel computers, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, eds. J. Dongarra, P. Kacsuk, and N. Podhorszki; volume 1908 of *Lecture Notes in Computer Science*, 200–207, Springer, 2000.
  - [15] 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.
  - [16] Kielmann, T., Hofman, R. F.H., Bal, H. E., Plaat, A., and Bhoedjang, R.A.F., MagPle: MPI collective communication operations for clustered wide area systems, Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99), pp. 131-140, Atlanta, 1999. Available from <http://www.cs.vu.nl/albatross/>.
  - [17] <http://www.lam-mpi.org/>.
  - [18] <http://www.epcc.ed.ac.uk/sun>
  - [19] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, version 1.1, University of Tennessee, 1995. Available from <http://www-unix.mcs.anl.gov/mpi/>.
  - [20] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface, University of Tennessee, 1997. Available from <http://www-unix.mcs.anl.gov/mpi/>.
  - [21] Gropp, W., Lusk, E. Doss, N., and Skjellum, A., A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, **22**, 789-828, 1996. Available from <http://www-unix.mcs.anl.gov/mpi/mpich/>.
  - [22] <http://www3.niu.edu/mpi/>.
  - [23] MPICH-GX: <http://www.moredream.org/mpich.htm>
  - [24] Park, K., Park, S., Kwon, O. and Park, H., A Private-IP-enabled MPI over Grid Environments , to appear in The 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA 2004) (Lecture Notes in Computer Science), December 2004.
  - [25] <http://www.open-mpi.org/>.
  - [26] <http://www.osl.iu.edu/research/impi/>.
  - [27] <http://www.pallas.com/>.
  - [28] <http://www.hlrs.de/organization/pds/projects/pacx-mpi/>.
  - [29] [http://www.csm.ornl.gov/pvm/pvm\\\_home.html](http://www.csm.ornl.gov/pvm/pvm\_home.html)

- 
- [30] Springel, V., User's Guide for GADGET Parallel Version 1.1. Available from <http://www.mpa-garching.mpg.de/galform/gadget/>.
- [31] Pers. Comm., Springel, V., Brief guide for GADGET-2.
- [32] Springel, V., Yoshida, N., White, S.D.M., GADGET: a code for collisionless and gasdynamical cosmological simulations, *New Astronomy*, **6**, 79-117 (2001).
- [33] Squyres, J.M., Lumsdaine, A., George, W.L., Hagedorn, J.G., and Devaney, J.E., The Interoperable Message Passing Interface (IPMI) Extensions to LAM/MPI, Available from <http://www.lam-mpi.org/papers/mpidc2000/>.
- [34] Thain, D., Tannenbaum, T., and Livny, M., Distributed Computing in Practice: The Condor Experience, Concurrency and Computation: Practice and Experience, to appear in 2004. Available from <http://www.cs.wisc.edu/condor/publications.html>.

### 1.3 List of Acronyms and Abbreviations

We list here a number of important acronyms used throughout the report, and references thereto.

IMPI	Interoperable MPI	<a href="http://impi.nist.gov/">http://impi.nist.gov/</a>
LAM MPI	Local Area Multicomputer Message Passing Interface	<a href="http://www.lam-mpi.org/">http://www.lam-mpi.org/</a> <a href="http://www-unix.mcs.anl.gov/mpi/">http://www-unix.mcs.anl.gov/mpi/</a>
MPICH	MPI Chameleon	<a href="http://www-unix.mcs.anl.gov/mpi/mpich/">http://www-unix.mcs.anl.gov/mpi/mpich/</a>
MPICH-G2	MPICH-Globus 2	<a href="http://www3.niu.edu/mpi/">http://www3.niu.edu/mpi/</a>
PACX	Parallel Computer Extension	<a href="http://www.hlrs.de/organization/pds/projects/pacx-mpi">http://www.hlrs.de/organization/pds/projects/pacx-mpi</a>

## 2. Grid MPI

### 2.1 Overview

The MPI standards [19] [20] have gained widespread success and acceptance in parallel programming and high performance computing (HPC). MPI provides effective and portable access to message passing functionality with which users can parallelise their own applications: GADGET 2.0 being a good example of this. All serious vendors of parallel machines provide their own native MPI implementations (although some MPI-2 features are less well supported), and there are a number of well-established tools available (e.g., TotalView [6], Vampir [27]) for debugging and profiling of MPI-based codes.

In the light of this success, it is then natural to consider using a Grid enabled version of MPI as the basis for porting the application to work in a Grid environment [8]. Here, users would not be limited to using a single parallel machine, but would be allowed to run MPI applications across several distributed resources in different administrative domains. The Grid-enabled MPI will deal with most of the infrastructure requirements transparently. However, some care is required to maximise performance when operating in such an environment.

An important point to remember at the outset is the type of problem for which Grid-enabled MPI might be useful, and in what context it might be best employed. Many problems arising in the physical sciences that have traditionally been the preserve of high-performance parallel computing are strongly coupled, that is, information from one part of the system propagates rapidly throughout the system (often requiring frequent global communication). In such an instance there is little to be gained in decomposing such a problem merely for the sake of running on a Grid. Such problems are probably best tackled on dedicated machines if sufficient capacity is available. However, if sheer scale of memory or CPU requirement demand distributed resources, then Grid resources are clearly one way to proceed.

More promising are weakly coupled problems or instances where one can make suitable approximations in strongly coupled problems. Here, communication between the distributed parts of the problem would be expected to be both less frequent and have smaller payload than the messages required for a strongly coupled problem. In this way, poor inter-machine latency and/or bandwidth would be less likely to have a major impact on performance. One could then imagine running comparable portions of a given problem on different machines that would require only infrequent synchronisation or communication. This mode of operation might be particularly appealing if a subset of the (perhaps heterogeneous) resources were available which, in combination, could be used to study larger problems than is practical on one resource alone. However, severe structural load imbalance between machines would suggest that there is little to be gained from using a Grid in this manner for such a system - the lesser part should merely be combined with the greater, thus saving inter-machine communication.

In the limit of negligible coupling within a problem, distributed high throughput systems such as Condor [34] have proved successful.

One useful way to look at Grid-enabled MPI is as a harness to bring together disparate tasks currently undertaken as a series of independent sub-problems on different machines between which data must be moved. Off-quoted examples here include real-time visualisation of simulation data and computational steering [4]. In the former, there is a strong argument that there are two weakly coupled tasks that should run on different hardware; in the latter, a Grid-enabled MPI could accomplish steering in a natural way that currently requires considerable additional code (generally in combination with linkable libraries [4] and/or the explicit use of sockets). It is in such integration of pre- and post-processing steps that make up part of the computational cycle that Grid-enabled MPI may find important use.

It is the aim of this document to provide an overview of the current state of technology and practice in Grid-enabled MPI, with a view to allowing the cosmological simulation code, GADGET2.0, to run over multiple sites of the DEISA infrastructure.

In addressing this issue, it is important not to become overburdened with many of the practical issues involved in using nascent Grid infrastructure (which cause a good deal of pain and consternation at present). So, while acknowledging their critical importance, details of Grid computing such as authentication, executable staging, process management, co-scheduling, remote file access and so forth are considered beyond the scope of this report.

Instead, this report concentrates on what functionality is offered to an MPI application programmer in order to allow him to execute his task. In order to be able to do this, one must consider factors such as the need for code modification, ease of compilation and use. In addition, the availability of support, tools, and relevant documentation is important. It is these issues that will ultimately decide which middleware, if any, is adopted most widely.

### 2.1.1 *Practical issues*

At the time of writing, a modest DEISA infrastructure is available, so only an overview of what middleware is currently in use or under development is given. In consequence, one can make little critical evaluation of the options. As the DEISA infrastructure becomes available, the study will be extended to cover a number of realistic cosmological simulations using GADGET2.0 with different middleware and how this performs. Only at this stage will critical evaluation become possible.

Work on the identification of challenging cosmological problems suitable for GADGET2.0 [31] are underway and are independent of any middleware issues at this stage. This is one advantage of using a standards-based approach that is independent of middleware.

### 2.1.2 *Document overview*

The following section covers currently available Grid-enabled MPI implementations and a number of related initiatives. The final section contains conclusions and directions for further work.

---

## 2.2 Standards

### 2.2.1 Interoperable MPI

While not an MPI implementation in itself, the Interoperable MPI (IMPI) [9], fronted by the US National Institute for Standards and Technology (NIST), is worthy of mention. It also serves to identify clearly a number of features that are desirable for the application programmer and so should also be featured in actual implementations.

The IMPI initiative was an attempt by (initially US) vendors and academic institutions with an interest in MPI to define a protocol through which different implementations of MPI would be able to communicate and hence interoperate. In this way the user could run an application across two or more machines each using a potentially different local native MPI implementation. This would clearly offer advantages over running a poorly optimised generic MPI on both machines. The IMPI protocol addresses a number of different issues, which are of varying degrees of interest to the user, and are used in this report as base criteria for a quality Grid-enabled MPI implementation.

- ? The protocol makes no additions, deletions, or changes to the standard MPI interface (the library API), so code retains portability. A correct MPI program must run correctly under IMPI. This is clearly an important issue for users: no alterations would be required to an existing code to run successfully (if not efficiently).
- ? Start-up and shutdown: an important task of any cross vendor MPI application running over multiple machines, and this applies equally well to Grid-enabled MPIs, is the negotiation of connections between the machines making up the distributed environment at the start of a job. IMPI achieves this via a single IMPI *server* that bootstraps communication between participating systems. IMPI implementations can then proceed to exchange messages. For the user, this process should be as simple as possible; for example, in the LAM implementation [17] of IMPI this appears as `impirun` instead of `mpirun`.
- ? Information about machine and network topology should be available to the user. IMPI achieves this dynamically via MPI communicator attributes. This allows the user to optimise the performance of an application at the cost of additional programming overhead.
- ? Security: IMPI implements authentication via the exchange of a key chosen by the user at start-up time. We note that HPC users are not always overly concerned with security of their data or computations. Any significant extra effort required here by the user might therefore be considered burdensome.

At present LAM/MPI [17] and the Hewlett-Packard implementation of MPI are known to support IMPI compliance and have provision for an IMPI server.

## 2.3 Grid-Enabled MPI Implementations

### 2.3.1 MPICH-G2

MPICH-G2 [15] is developed at Northern Illinois University [22] and is an implementation of the MPI 1.1 standard build upon MPICH [21]. It makes use of the Globus Toolkit version 2 (GT2) [10]. In particular, it implements the so-called

*globus2 device* for MPICH by using GT2 to provide services such as authentication, authorisation, remote I/O and control of remote processes. Use of the Globus Toolkit allows MPICH-G2 to hide some of the complexity of a heterogeneous environment from the user, who must merely be in possession of a valid X.509 proxy certificate to run MPI jobs. The MPICH-G2 `mpirun` command then takes care of submitting the job, staging the executable, and so forth. At the most basic level, no alterations to user code are required to run MPICH-G2.

Communications in MPICH-G2 are split into different types: those using TCP and those using the underlying vendor-supplied MPI implementation. TCP communications are split further into wide area, local area, and system area communications. Wide area communications are those between two different sites, local area between two different machines at the same site, and system area communications are those between different compute nodes of a cluster or even between different processes on the same machine. MPICH-G2 is aware of these different communication types, describing them via topology *depths* and *colours*, which are used to provide topology aware collective operations that the programmer can use to optimise communications. For example, processors that can only communicate via a wide area network are assigned different topology depths. This topology information is also available to application programmers via `MPI_Attr_get()` to enable optimisation by matching decomposition to network topology.

The following code illustrates a possible use of topology information obtained dynamically via MPICH-G2 (based on [15]).

```
#include <mpi.h>

int main (int argc, char ** argv) {

    int me, flag;
    int * depths;
    int ** colours;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
    MPI_Attr_get(MPI_COMM_WORLD, MPICHX_TOPOLOGY_DEPTHS, &depths, &flag);
    MPI_Attr_get(MPI_COMM_WORLD, MPICHX_TOPOLOGY_COLORS, &colours, &flag);

    /* Code is now able to split communicators based on the
     * information in topology depths and colours */
    ...

    MPI_finalize();
}
```

There are a number of practical disadvantages inherent in the MPICH-G2 approach. First, every compute node must have a public IP address and a large number of ports need to be opened in any relevant network firewall to allow inter-machine communications. Second, jobs not started at approximately the same time on all machines may cause problems. However, there are advantages in the single sign on approach inherited from the Globus Toolkit.

Code and documentation are available and are under active development.

### 2.3.1.1 Related projects

A number of projects based on MPICH-G2 are underway which attempt to address shortcomings in the original. Two of these projects are based in South Korea (MPICH-GX [23] and MPICH-GP [24]) and aim to solve problems with private IP addresses and address quality-of-service issues.

### 2.3.2 LAM/MPI

LAM/MPI (Local Area Multicomputer/MPI) is a well-established implementation offering a complete set of MPI 1.2 features and a subset of MPI 2 features [1], [17]. LAM/MPI is developed at Indiana University and the current public release is 7.1. The normal mode of operation for LAM/MPI in a local cluster environment is to set up a *host file* containing the network addresses of the participating resources, and then to launch the LAM runtime environment via the `lamboot` command to start the appropriate daemon on each of the required machines. An MPI job would then be started via `mpirun` that, in the cluster environment, would use `ssh` or `rsh` to launch remote executables. As this approach would not extend to a Grid environment, LAM/MPI 7.1 includes two developments related to Grid operation.

First, LAM/MPI provides a limited support for execution within a Globus Toolkit 2 (GT2) [10] environment via a boot module that obtains the necessary LAM daemons via the Globus Resource Allocation Manager (GRAM) and `globus-job-run`. In this case, the *host file* is retained and details the required resources as GT2 contact strings. While this requires no alteration of existing code, there appears to be no mechanism for the user to recover information about network topology. Second, LAM/MPI provides support for IMPH (apart from some of the collective communication routines) [33] making use of an IMPH server developed at Indiana [26].

LAM/MPI is extensively documented and is under active development.

### 2.3.3 PACX-MPI

PACX-MPI (PARallel Computer eXtension MPI) is an implementation of MPI 1.2 and a subset of MPI 2.0 developed at the High Performance Computing Centre (HLRS) in Stuttgart, Germany [28]. It is targeted at traditional HPC resources, rather than clusters of workstations, and aims to use the existing communication system of each machine. No changes to a user's code are required.

PACX-MPI implements intra-machine communication using the native vendor MPI, while inter-machine communications employ TCP communication. Such TCP communications can use the secure socket layer (SSL) for encryption, perhaps with negative performance implications. Two additional MPI processes are started on each different machine to mediate inter-machine communication, one responsible for input and one for output. This approach has the advantage of restricting the number of open ports needed in any relevant firewall. Unfortunately, the node on which these communication processes are started still requires a public IP address, although the fact that only two processes are involved may simplify the use of a proxy for port forwarding.

Launching a multi-site job with PACX-MPI depends on the network configuration of the machines involved. If the network addresses are known then one `mpirun` command is used to start the job on all machines. Alternatively, the job may be

---

started at each site in turn, each machine waiting for the others to connect. This can complicate the launch process and make it potentially more expensive to run as computing cycles will be wasted.

Advantages of PACX-MPI are the ability to use it with no change to existing code, and the small number of TCP connections used for communications between sites.

#### 2.3.4 *StaMPI*

StaMPI is an MPI implementation from the Centre for Promotion of Computational Science and Engineering (CCSE) of the Japanese Atomic Energy Research Institute (JAERI) providing MPI 1.2 and a subset of MPI 2.0, [14]. A StaMPI-I/O library implementation based on MPI-2 I/O is also available. The software is available on request from [3].

StaMPI operates by allowing static or dynamic process creation (based on the MPI-2 routine `MPI_Comm_spawn()`). This potentially allows the user significant flexibility in code to control communication topology. In practice, however, many users do not want to do this. StaMPI utilises vendor MPI for intra-machine communication and TCP/IP for inter-machine communication.

Documentation for users appears to be available only in Japanese.

#### 2.3.5 *Related issues*

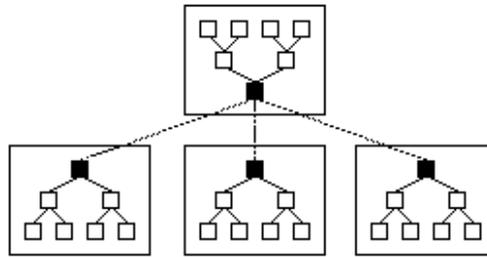
##### 2.3.5.1 *Open MPI*

The Open MPI project [25] is a recent development which aims to build on a number of existing implementations (including LAM/MPI and PACX-MPI) to produce a high-quality open-source MPI-2 implementation. The first products are expected in the first quarter of 2005.

##### 2.3.5.2 *MagPle*

MagPle is a library built on top of MPICH that provides collective communication that is aware of network topology [16]. MagPle is developed by a Dutch consortium based at Vrije University in Amsterdam. The latest version appears to be 2.0, [5].

For a given network topology, MagPle attempts to minimise both the number and size of messages sent over high latency/poor bandwidth connections (see Fig. 1). The strategy makes use of the hierarchical nature of many MPI collectives, which are often implemented via tree algorithms. MagPle draws the distinction between asymmetric collective operations, such as `MPI_Bcast`, where there is a single originating or receiving (i.e. the root process) and symmetric operations such as `MPI_Alltoall`. The later clearly require a larger number of inter-machine communications.



**Figure 1: Communication over a differentiated network topology with relatively poor latency and bandwidth between machines (dotted lines). For an asymmetric MPI collective such as `MPI_Bcast()`, inter-machine communication is minimised by using only a single processor on each machine (full squares). The same processors are responsible for coordinating intra-machine communication (assumed to be relatively fast). Redrawn from [16].**

MagPle has largely been superseded by developments at the MPI implementation level, and there appears to be no active development of the product.

### 2.3.5.3 PVM

While elements of PVM (Parallel Virtual Machine) are still relevant when considering Grid-enabled MPI, PVM has been largely abandoned by application programmers in favour of MPI. There appears to be little active development of the PVM product [29].

## 2.4 Summary

We have provided an overview of the current availability of Grid-enabled MPI. Of the libraries available, the recommendation from this is that at least MPICH-G2 and PACX-MPI should be made available on the core DEISA infrastructure. This will allow a comparison to be carried out, to determine which library is best suited for running GADGET 2.0 over the DEISA infrastructure. However, future developments in Open MPI should also be monitored.

## 3. GADGET

### 3.1 The GADGET Code

GADGET (GALaxies with Dark matter and Gas intEracT) is a freely available standard C code, distributed under the GNU General Public Licence Version 2, for the investigation of gravitational and cosmological problems [31]. The code essentially solves the gravitational  $N$ -body problem, but includes a range of options for extra physics. These are introduced via a subdivision of the  $N$  particles into different types: gas, halo, disk, bulge, stars, and boundary each of which may be subject to different physical processes. For example, the fluid dynamics of the gas particles are computed via Smoothed Particle Hydrodynamics (SPH); the remaining particles are all dark matter particles and, along with the gas particles, are subject to gravitational forces.

The code is a parallel code, where communication is achieved via the Message Passing Interface (MPI). The parallelisation strategy uses a domain decomposition in which

---

particles are distributed between processors depending on their spatial position in the simulation volume. As the position of a given particle evolves, responsibility for it might be passed between processors. Load balancing is achieved via orthogonal recursive bisection of the simulation volume: this process is carried out dynamically so that load remains balanced as structure evolves within the system, becoming more and more clustered.

The code uses a number of sophisticated algorithms to address the N-body  $O(N^2)$  scaling problem for large numbers of particles. Long-range gravitational forces are computed via a Particle-Mesh (PM) method [31] employing discrete Fourier transforms. Computation of the short-range component of the gravitational interactions is optimised using a Barnes-Hut [1] tree code algorithm, or *Treecode*. Explicit interactions may be required between particles located in different processor's sub-domains hence some particle information must be exchanged between processors via message passing.

The version of the code considered for this work is GADGET 2.0<sup>1</sup> [31] obtained from Volker Springel of the Max Planck Institute for Astrophysics in Garching.

### 3.2 **Compilation and Testing**

The standard GADGET 2.0 code as provided by Volker Springel has been compiled and tested on a number of different target platforms. These include the IBM p690+ cluster (HPCx [13]) and a Sun E15000, 52-way SMP platform (Lomond [18]) at EPCC. Testing has been carried out against a standard problem described in the GADGET 1.1 documentation [30] involving 262,144 gas particles and 262,144 halo particles (i.e. a modestly sized system). A correct final result data set and timing for this problem has been provided by Adrian Jenkins, Durham University, of the Virgo Consortium. This will be referred to as the Durham test data set.

GADGET 2.0 requires a small number of third-party publicly available libraries including the Gnu Scientific Library (GSL, [11]) and the Fastest Fourier Transform in the West (FFTW, [7]). These were compiled and installed where necessary.

Compilation options on HPCx were taken in line with recommendation and experience (specifically `mpcc_r -q64 -qipa -qhot -O5 -qstrict`).

### 3.3 **Performance on HPCx**

#### 3.3.1 *The qsort routine*

Profiling of the standard code on a single node (32 processors with shared memory) to assess the code's base performance. Initial results for timing are shown in Fig 2 for the current AIX 5.2 on HPCx.

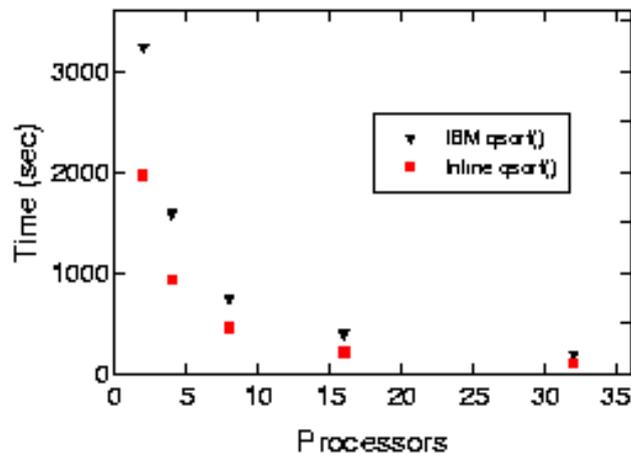
Investigation revealed a problem with the IBM C compiler: it failed (even with an explicit request `-qinline+domain_compare_key`) to inline comparison functions provided in the standard C library for `qsort()`. Specifically, this was related to the function `domain_compare_key()` in the dynamic load balancing section of the code (and to a lesser extent in other invocations of `qsort`). The problem was resolved by providing a

---

<sup>1</sup> GADGET 2.0 is currently not publicly available.

separate `qsort()` function, in source form, with explicit inlining of the comparison function. Overall performance on the Durham test data set was improved by roughly a factor of 2 (see Fig. 2). This may be an extreme case, as the dynamic load balancing is called at every step in the early stages of the calculation.

A similar performance problem has been noticed by Volker Springel and was attributed to the `qsort()` algorithm provided by IBM which exhibits poor behaviour if the input is already sorted. The problem may be dependent on AIX version (currently AIX2.5 on HPCx).



**Figure 2: Graph showing performance of the GADGET 2.0 code on HPCx for two cases in which the IBM standard C library `qsort()` function is replaced by an alternative. Performance is improved by up to a factor of two.**

### 3.3.2 MPI Performance

For the improved version, where `qsort` has been updated, one can assess the profile of the MPI usage from the following based on 32 time steps on 32 processors.

MPI Routine	#calls	avg. bytes	time(sec)
MPI_Comm_size	5	0.0	0.000
MPI_Comm_rank	5	0.0	0.000
MPI_Send	341	5.8	0.001
MPI_Ssend	959	19186.1	0.150
MPI_Recv	1026	19990.9	0.195
MPI_Sendrecv	8263	16407.7	0.999
MPI_Bcast	71	110.7	0.029
MPI_Barrier	171	0.0	8.547
MPI_Gather	204	6.7	0.020
MPI_Allgather	781	38.8	4.395
MPI_Allgatherv	34	19487.5	0.049
MPI_Reduce	310	9.9	0.023
MPI_Allreduce	342	6436.5	0.063
MPI_Alltoall	66	16640.0	0.149

---

-----

The largest number of calls is to `MPI_Sendrecv()` to exchange active particle information between sub-domains for force, hydrodynamic, and density calculations. A measure of load imbalance is indicated by the large amount of time spent in `MPI_Barrier()`. Further load imbalance is indicated by the length of time spent in `MPI_Allgather()`, as this routine, like the other collective communication routines in MPI, contains an implicit barrier.

## 4. Grid-enabling GADGET

An initial investigation suggests that running GADGET 2.0 using the Durham test data set is a strongly coupled problem that requires frequent global communication. However, the code is designed to be as efficient as possible for large problems so there is clear scope for Grid operation for scaling purposes. Furthermore, upon deeper investigation, there does appear to be possibilities for allocating sections of the computation to different supercomputers within the DEISA infrastructure thus increasing the resources that can be used to tackle bigger classes of problems or existing problem sizes more rapidly.

### 4.1 *Meta-computing*

After porting GADGET 2.0 to the DEISA infrastructure, the code will be run over multiple supercomputing platforms residing at geographically separated sites. In the first instance, the expectation is that the code will perform poorly, as a code naively run in a meta-computing environment, i.e. employing more than one geographically separated supercomputing sites, without taking into account the different communication latencies involved may involve a lot of unnecessary high-latency communications between sites. Thus, attempts will be made to find the best decomposition over different sites. This process of separating computational workloads, where each load will run on separate sites, will be referred to as *decoupling*.

Once routines have been recognised as potential candidates for decoupling, then new MPI communicators will be introduced to the code. Each decoupled routine will have its own MPI communicator. (Currently, GADGET 2.0 used a single MPI communicator throughout.) Processors will then be made aware of their location within the framework via their MPI communicator using `MPI_Comm_get_attr()`. One idea is that processors are then renumbered *before* data is loaded, which is replicated and staged at all the participating sites in advance.

There are a number of code sections, that appear to be mutually exclusive, that may be candidates for decoupling and will be investigated.

#### 4.1.1 *Decoupling SPH from the gravitational computation.*

As described in Section 3.1, the code has two sections that are run each time step, namely the gravitational N-body problem and the evolution of the gas. These two sections can be decoupled as they can be evaluated independently of each other and thus involve communications only once every time step, so that SPH runs on one site whilst the gravitational section (Treecode+PM) runs on another.

#### 4.1.2 *Decoupling the Treecode and the PM sections.*

The gravitational section can also be decoupled by separating the Treecode from the PM computation. (The PM section involves an FFT that communicates over all participating processors, thus GADGET 2.0 would be forced to run the PM within one site.) The border between long-range and short-range computation can be moved, by simply making the PM grid courses, so that work can be shifted from the Treecode to the PM section instead. Thus, the code could be dynamically tuned to have a balanced workload over the two supercomputer sites. Lastly, it should also be noted that the PM section could be run once every 10 time steps, say.

#### 4.1.3 *Decoupling the sub-trees of the Treecode.*

It appears that it is possible to decouple the separate Treecode computation, thus we can spread the Treecode computation across sites. However, the effectiveness of this scheme remains to be seen, as there is communication between sub-trees.

### **4.2 *Single Site Optimisations***

The performance study will be extended to larger problems on a single machine to gain a fuller understanding of the issues involved in scaling to very large problems.

#### 4.2.1 *Self Tuning*

One method of improving the performance of GADGET 2.0 is to actively monitor the performance using the information in the dynamic domain decomposition to feedback and thus improve the dynamic load balancing algorithms. By introducing routines to measure actual execution times, the code can alter the future domain decomposition based on past experience. This might be particularly important on heterogeneous Grids. Indeed, the experience gained here (with Gadget 2.0) can be applied to other applications.

#### 4.2.2 *Optimising Communications*

The code itself may benefit from optimisations of the communication routines, as described in the following subsections.

##### 4.2.2.1 *Collective Communications*

Clearly, given the number of collective communication events in the code, it will be important to use optimised collective MPI routines. Here, we are reliant largely on the MPI implementation. However, there may be scope for elimination or consolidation of existing collective operations.

##### 4.2.2.2 *Blocking versus Non-Blocking Communications*

Communication of particle information between processors takes place (three times per time step in different contexts) as an ordered series of `MPI_Sendrecv()` operations. The performance of these operations, while not problematic for the Durham test data set, may become so for larger problems. It may be necessary to revisit this area, e.g., to replace `MPI_Sendrecv()` with explicit non-blocking communications, which have been shown to out-perform blocking communications on HPCx [12].

#### 4.2.2.3 *Removing Barriers*

As was seen in Section 0, the code spends a lot of time in `MPI_Barrier`. This could simply be a symptom of a poor load balance, however, some of these calls may be superfluous and might be eliminated.

### 4.3 *Homogeneous and Zoom Simulations*

GADGET 2.0 is used to study two basic types of problem: *homogenous* problems in which particles are distributed in a broadly uniform fashion throughout the simulation space, and so-called *Zoom* problems, in which a smaller, high-resolution region of a large cosmological simulation, is re-simulated at a higher resolution.

The ideas expressed in Sections 4.1 and 4.2 can be employed for both the homogenous and Zoom type simulations. However, it should be noted that Zoom simulations are very difficult to load balance even on a single machine, so efforts on the Grid side will initially concentrate on homogeneous type of problems.

However, one distinct possibility for decoupling lies with Zoom simulations, where a number of these simulations can be run concurrently on a number of different sites, where communication only occurs at the start of the simulation. These Zoom simulations are, therefore, weakly coupled and should perform well over the DEISA infrastructure. New code will have to be introduced to GADGET 2.0 to allow such a farming out of simulations to be possible. Of course, it may be that the most efficient use of the DEISA infrastructure for Zoom simulations would be to submit multiple jobs to the batch system by hand.

## 5. Conclusion and Future Work

This document has described our investigation into the current state of Grid-enabled MPI and our initial experiences with porting and running GADGET 2.0. The remainder of this work package will continue as follows.

### 5.1 *Comparing PACX-MPI and MPICH-G2*

Perform an investigation comparing PACX-MPI and MPICH-G2 using a simple MPI code on the DEISA infrastructure and note any constraints that arise through their usage. If these Grid-enabled MPI libraries are not available in the infrastructure, then a mini-Grid employing both HPCx (IBM p690+ cluster) and Lomond (Sun E15000, 52-way SMP) will be utilised.

### 5.2 *Porting GADGET*

GADGET 2.0 will be ported to the available Grid, either the DEISA core-infrastructure or the mini-Grid, along with the necessary data files.

### 5.3 *Tuning GADGET*

GADGET 2.0 will be run over multiple sites.

Then, new MPI communicators may be introduced into GADGET 2.0, and the code made topologically aware, thus allowing for processor renumbering depending on their own location.

Different methods of decoupling will be investigated, including

- 1/ SPH from Gravitational computation,
- 2/ the Treecode from the PM section,
- 3/ the sub-trees within the Treecode.

#### **5.4 Zoom simulations**

If the attempts at decoupling, as described in 5.3, are not as successful as hoped, then running the Zoom simulations as a farm will be considered, although this requires new code to be written.

#### **5.5 Optimising Communication Routines**

Communication routines will be reviewed with a view to consolidating existing calls to MPI routines and/or rewrite the routines anew.

#### **5.6 Introducing Self-Tuning**

If time permits, then the idea of introducing self-tuning will be considered, to allow a more efficient method of domain decomposition.