



CONTRACT NUMBER 508830

DEISA
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR
SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

Grid-enabled implementation of GADGET
for metacomputing environments

Deliverable ID: D-JRA2-1.2
Due date: October, 31, 2005
Actual delivery date: November 28, 2005
Lead contractor for this deliverable: EPCC, UK

Project start date: May 1st, 2004
Duration: 4 years

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

Document Keywords and Abstract

Keywords:	Virgo, Cosmology, GADGET, Grid, Message Passing Interface, MPI, PACX-MPI, MPICH-G2, meta-computing
Abstract:	<p>This document describes the work undertaken to allow the Cosmological simulation code GADGET2 to be run using Grid enabled message passing via the PACX MPI or MPICH-G2 libraries.</p> <p>Basic information on MPI latencies and bandwidths can be used, along with information about code performance and scaling, to estimate under what circumstances it may be viable to run the GADGET2 code between two different machines comprising a meta-computer. These estimates are confirmed with results from a test system in Edinburgh.</p> <p>The DEISA grid infrastructure at RZG and IDRIS, along with the DEISA network, has been used to run the GADGET code, although the results suggest that this kind of meta-computing is not viable with current technology.</p> <p>It is demonstrated that the code written to perform the domain decomposition in the grid context incurs no overhead compared with the standard GADGET domain decomposition code. However, attempts at dynamic load balancing using measured CPU time per process were unsuccessful in improving code performance.</p>

Table of Content

1 INTRODUCTION	2
1.1 Executive Summary	2
1.2 References and Applicable Documents.....	2
1.3 List of Acronyms and Abbreviations	3
2 AN OVERVIEW OF GADGET2.....	4
2.1 Large Scale Cosmology.....	4
2.2 Parallel Load Balancing	5
2.3 Scaling.....	6
2.4 The Tree vs Particle-Mesh Trade-Off	7
3 GRIDS AND GRID MPI	8
3.1 Available Grids.....	9
3.1.1 Machine Specifications.....	9
3.1.2 MPI Latency and Bandwidth.....	9
3.2 GADGET on the Grid.....	10
3.2.1 Homogeneous Cluster: SUN e3500 and e15000	10
3.2.2 Heterogeneous Cluster: SUN e15000 and IBM p690.....	12
3.2.3 DEISA Infrastructure.....	12
3.3 General Comments.....	12
4 LOAD BALANCING	14
4.1 The Domain Decomposition	14
4.1.1 Standard Decomposition Code.....	14
4.1.2 DEISA Decomposition Code	14
4.2 Dynamic Load Balancing	16
5 CONCLUSIONS	17

1 Introduction

1.1 *Executive Summary*

The objective of this work was to evaluate and implement then necessary changes to enable GADGET, a large MPI cosmology simulation, to run using multiple sites of the DEISA infrastructure in order to perform much larger simulations.

We have enabled the GADGET code to run in a grid environment using the PACX MPI and MPICH-G2 libraries. We note that documents D-JRA2-1.1 and D-JRA2-1.3 discuss, respectively, the background and practical issues involved. This document is concerned with the performance and scaling of the code in a grid context.

One of the essential issues in parallel code is to maintain a good communication to computation ratio. MPI latency and bandwidth is measured for various native and grid-enabled MPIs and the implications for the performance of the GADGET code are discussed. We demonstrate on a small test grid that if the communication to computation ratio can be maintained at a reasonable level, i.e., if the problem size is large enough, then grid-enabled MPI can be viable.

The DEISA grid infrastructure at RZG and IDRIS, along with the DEISA network, has been used to run the GADGET code, although the results suggest that the overheads are too high to be viable with available technology. We conclude that grid-enabled MPI is not currently useful for GADGET in the DEISA context.

In addition, the DEISA infrastructure would have to provide co-scheduling between participating sites if grid-enabled MPI is to be a viable tool for production use for any purpose.

The main results are as follows. The grid-enabled version of GADGET is as good as the original form, which allows the original objective to be achieved if the underlying infrastructure is suitable. The performance of the grid-enabled code on DEISA is below expectation, due to bandwidth and latency. The new load balancing strategy does not improved performance.

In conclusion, to use the grid-enabled version of GADGET2 in a metacomputing environment, one requires an extreme problem size and the ability to co-schedule tasks.

Finally, for the near future, the present DEISA strategy to run the complete application on the largest available system is sound.

1.2 *References and Applicable Documents*

- [1] Appel, A.W., SIAM J. Sci. Stat. Comp., 6, 85 (1985).
- [2] Barnes, J., and P. Hut, Nature, 324, 446 (1986).
- [3] Capuzzo-Dolcetta, R., Comput. Phys. Commun., 169, 365-369 (2005); 10.1016/j.cpc.2005.03.081.
- [4] Couchman, H.M.P., ApJ, 368, 23 (1991).

- [5] D-JRA2-1.1: Specifications document for D-JRA2-1.2
- [6] D-JRA2-1.3 Documentation for D-JRA2-1.2
- [7] Efstathiou, G., M. Davis, C.S. Frenk, and S.D.M. White, *ApJS*, 57, 241 (1985).
- [8] Hockney, R.W., and J.W. Eastwood, *Computer Simulation Using Particles*, Institute of Physics Publishing, Bristol, (1988).
- [9] Karonis, N.T., B. Toonen, and I. Foster, *J. Parallel and Distributed computing*, 63, 551 (2003).
- [10] Lucy, L.B., *Astrophys. J.*, 82, 1013 (1977).
- [11] PACX-MPI. See <http://www.hlrs.de/organization/pds/projects/pacx-mpi/>.
- [12] Sloan Digital Sky Survey. See <http://www.sdss.org/>.
- [13] Springel, V., N. Yoshida, and S.D.M. White, *New Astronomy*, 6, 79 (2001).
- [14] Springel, V., in preparation (2005).
- [15] Springel, V, et al., *Nature*, 435, 629 (2005).
- [16] UK nation supercomputing service <http://www.hpcx.ac.uk/>.
- [17] Xu, G., *ApJS*, 98, 355 (1995).

1.3 List of Acronyms and Abbreviations

GADGET	GA laxies with D ark matter and G as intErac T
MPI	M essage P assing I nterface
PACX	P arallel C omputer E Xtension
MPICH-G2	M PI C Hameleon G lobus 2
PM	P article M esh
SPH	S moothed P article H ydrodynamics

2 An Overview of GADGET2

2.1 Large Scale Cosmology

The formation of large scale structure in the universe in the aftermath of the Big Bang has become a major theme in modern cosmology. Large scale observational studies such as the Sloan Digital Sky Survey [12] have revealed an unprecedented level of details in the distribution of galaxies at many different length scales in the visible universe (from clusters of individual galaxies at scales of around 10^7 light years to the largest observed structures at $10^8 - 10^9$ light years). Computational tools which allow cosmologists to investigate how such a spectrum of structure might come about are therefore invaluable in complementing theoretical and observational work.

However, the computational challenge presented by the gravitational N-body problem at the heart of cosmology is formidable. The long-range nature of the gravitational force means that the cost of a direct calculation scales as $O(N^2)$, where N is the number of particles involved. For collisional dynamics, where the characteristic time between particle collisions is small compared to the crossing time of the system under investigation, a full N-body treatment can be required to achieve acceptable accuracy [3]. An example of such a collisional system is a star cluster, in which the number of simulation particles might be limited to perhaps $10^6 - 10^9$. Fortunately, on the large scale it can be assumed that dynamics is collisionless, while forces between particles at the shortest length scales are smoothed in some way. This makes a number of alternative methods available which reduce the computational scaling to a more tractable $O(N \log N)$.

One commonly used approach is based on highly efficient Fourier techniques, broadly referred to as particle-mesh (PM) methods (see, for example, [8]). While PM methods are very good at dealing with long range interactions, forces between particles at the mesh scale are not handled well. This means that something else is needed to represent dynamics in regions where resolution greater than the mesh scale is required. This “something else” might involve direct calculation, i.e., a particle-particle particle-mesh algorithm (often referred to as P3M [7]), or adaptive mesh refinement to capture structure in the regions requiring high resolution [4].

A second approach is that of the hierarchical multipole expansion, or tree code [1]. Here, particles are divided into groups depending on their spatial position within the simulation volume, usually on the basis of a Barnes-Hut [2] recursive oct-tree partitioning of the space. The total force on a given particle can then be approximated by computing interactions between near-neighbours directly, but computing long-range interactions via a multipole expansion involving distant groups. In the limit that the distant groups are better and better resolved (to the point where they contain a single particle) the exact force is essentially recovered. The advantage of the Tree approach is that, not being tied to any fixed mesh, it can capture highly inhomogeneous mass distributions efficiently and effectively.

To combine the huge dynamic range available to the Tree code and the efficiency of the Fourier based PM methods, the Tree Particle Mesh (TreePM) code has been developed [17]. This retains the Tree for short-range and intermediate-range interactions, while

resorting to the PM for the long-range forces. One such code is GADGET [13]: the acronym is derived from “Galaxies and Dark Matter and Gas interact”. At the time of writing GADGET has been used to run the largest ever cosmological simulation (the so-called Millennium Run [15]) involving 2160^3 particles. It is GADGET which is the focus of this work.

One further important physical consideration in cosmological simulations is the inclusion of gas, or hydrodynamics. As is the case for gravitational forces, hydrodynamics can be introduced either via a mesh-based (Eulerian) method or a particle-based (Lagrangian) method. GADGET adopts the latter in the form of Smoothed Particle Hydrodynamics (SPH [10]). This solves the continuous Navier-Stokes equations by locally reconstructing a continuous density field from the particle distribution via a spline method. Like the tree method for the gravitational problem, SPH has the advantage over mesh-based methods that it can capture hydrodynamics for very inhomogeneous mass distributions.

In the following sections, there is an overview of some of the features of the GADGET code which are pertinent to the present work. In particular, GADGET is designed to run on parallel computers, so the issues of scaling and load balance are very important. The balance of work done by the Tree part of the code and the PM part of the code is also examined via illustrative benchmarks.

2.2 Parallel Load Balancing

One of the key issues in the parallel implementation of a given algorithm is maintaining an even distribution of work between processors, i.e., maintaining a load balance. For a mesh-based problem this is usually a relatively straightforward problem – a regular geometric distribution of mesh points between processors will generally guarantee static load balance. For a non-mesh based method such as the Tree code, and one in which the distribution of work can change significantly during the calculation, the situation is more difficult.

A common solution in non-mesh based parallel codes is the use of domain decompositions based on space-filling fractal curves. GADGET uses a decomposition based on the Peano-Hilbert curve (see Figure 1). The simulation volume is divided recursively to generate the Peano-Hilbert curve, which has the useful property that points close together in 3-dimensional space tend to lie close together on the 1-dimensional curve. This means the 3-dimensional problem can be sub-divided into parts of arbitrary length on the Peano-Hilbert curve and always give rise to compact domains with low surface area to volume ratio.

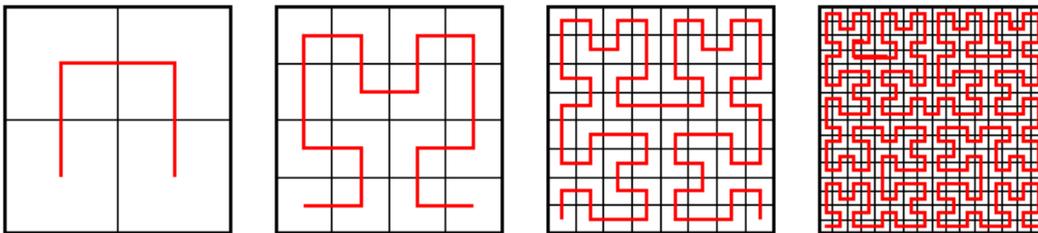


Figure 1. The Peano-Hilbert curve in two dimensions. The curve is generated by

recursively subdividing space and adding new sections of curve with the appropriate rotations. The generalisation to three dimensions is straightforward.

Particles are arranged along the Peano-Hilbert curve depending on their position in space. The Peano-Hilbert curve can then be divided into segments which contain equal amounts of work as measured simply by the number of particles, or a more suitable measure such as the total number of interactions computed for particles in that segment. The small surface area to volume ratio of the Peano-Hilbert segments ensures that the code minimises the communication overhead involved in passing particle information between processes.

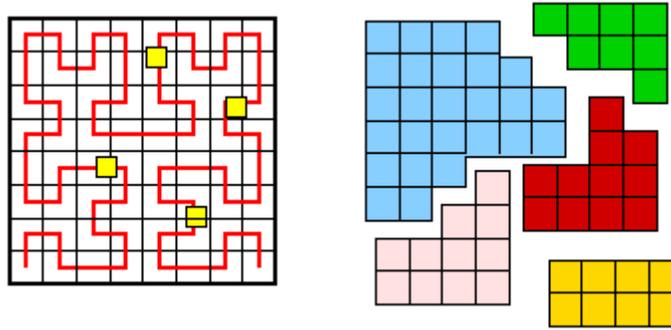


Figure 2. The Peano-Hilbert curve (here drawn in two dimensions) may be decomposed at arbitrary points along its length and still give rise to compact domains with low surface area to volume ratio in three dimensions. Redrawn from [14].

The Peano-Hilbert curve also has the important advantage that it is commensurate with the Barnes-Hut oct-tree decomposition used in GADGET.

One further important consideration is that of memory constraints, which may ultimately limit scope for load balance. The largest-scale cosmological simulations put an emphasis on size and number of particles, and so tend to be memory-limited. This means that it may not be possible to move work (i.e., particles) between processors for the sake of balancing work-load owing to a limit in the amount of memory. GADGET usually allows each processor some free memory (perhaps 50-100% of the total) to accommodate extra particles in order to balance the computational load. The “over-allocation” of memory per process is determined by the input parameter `PartAllocFactor` for particles.

2.3 Scaling

One of the most important issues for parallel codes is that of scaling, i.e., how the time to solution is reduced as a function of the number of processors used. As an illustrative example, Figure 3 shows the time taken in some different parts of the GADGET code for a given problem as a function of the number of processors on the HPCx system [16]. The times are measured relative to 32 processors (one IBM p690+ frame) which is the minimum unit of resource used for accounting purposes (i.e., all jobs should use at least this many processors). The problem size is fixed: 4,194,304 gas particles with the same particle mesh resolution in each case (PMGRID=128).

For this problem, the main parts of the calculation – the gravity tree and the hydrodynamics, and to a lesser extent the particle mesh - scale reasonably well. Even the load imbalance is reduced reasonably effectively by reducing the number of processors. However, the main obstacle to scalability here appears to be the domain decomposition, which scales adversely at 128 processors and above. The reason is that the domain decomposition code is essentially serial and limits scaling following a classic Amdahl's Law argument. (It should be noted that these times are based on a relatively short number of iterations during the start-up phase when the domain decomposition is called at very step. The frequency can be reduced as the simulation proceeds but further studies suggest the impact can still be significant for longer runs.)

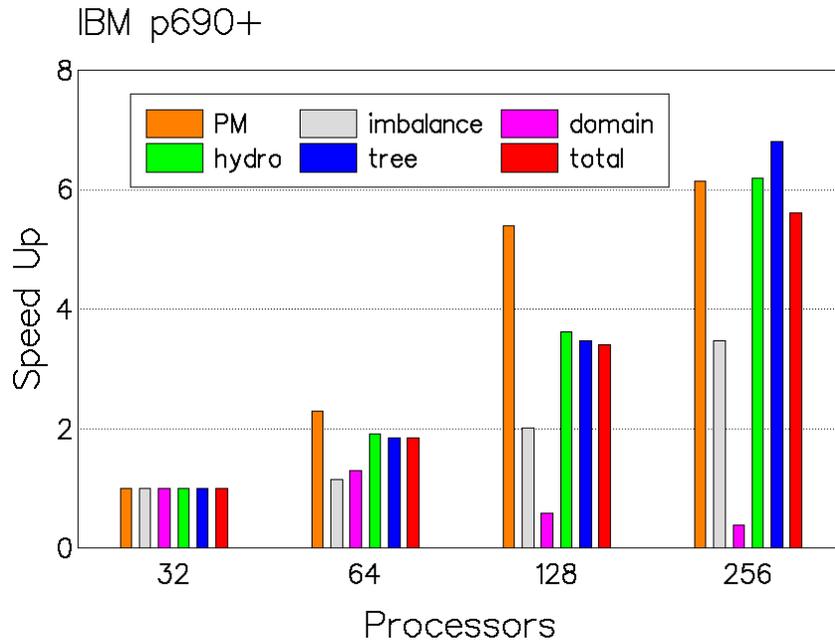


Figure 3. An example of scaling of the GADGET code on a p690+ system for fixed problem size and particle mesh resolution. The bars show the time taken in various parts of the computation measured relative to 32 processors: from left to right the bars indicate the particle mesh, load imbalance, domain decomposition, hydrodynamics, the gravity tree, and the total as measured by GADGET's internal timing.

In general, scaling can be improved by increasing the problem size. However, in order that the code be scalable to truly large numbers of processors, a number of alterations to the design would probably be required. For example, the code uses quite a large number of data structures with local size proportional to the number of MPI tasks, with associated use of global communications. Although leading to more complexity, these might have to be replaced in future.

2.4 The Tree vs Particle-Mesh Trade-Off

A further important consideration in the performance of the Tree-PM algorithm is the trade off between work done in the tree part and work done via the particle mesh. A

modest particle mesh resolution will increase the load on the relatively expensive tree calculation, while increased particle mesh resolution will move more work to the efficient particle mesh. However, there comes a point at which further increases in particle mesh resolution are of no additional benefit. This is illustrated in Figure 4, where for the same problem as above (4,194,304 gas particles) the particle mesh resolution has been varied on a fixed number of processors.

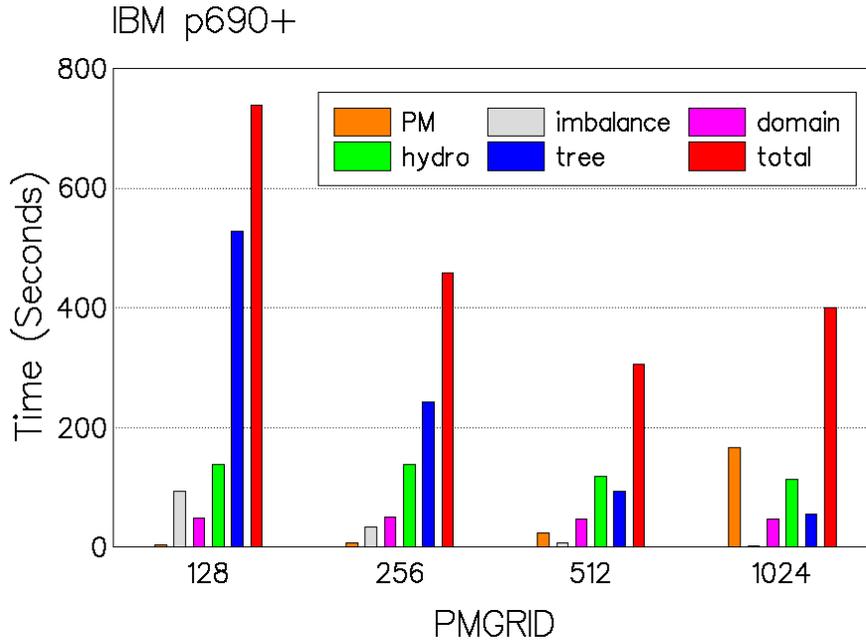


Figure 4. GADGET times for a fixed problem on a fixed number of processors as a function of the particle mesh resolution PMGRID. From left to right the bars indicate the particle mesh, load imbalance, domain decomposition, hydrodynamics, the gravity tree, and the total.

This trade-off could become more important in a grid context, where the collective communication required for the PM calculation would place higher stress on the inter-machine bandwidth than the point-to-point communications required for the tree calculation.

The particle mesh is essentially exact, so is probably to be preferred as long as it does not compromise the ability of the simulation to pick up the required high resolution features. GADGET also has the capability of inserting high-resolution “zoom” regions (essentially static mesh refinement).

3 Grids and Grid MPI

A number of machines have been used to perform the work reported here, including machines at IDRIS and RZG Garching which form part of the DEISA infrastructure. Two grid-enabled MPI libraries are considered: PACX-MPI version 4.1.4 [11] and MPICH-G2

[9]. Operational details for PACX-MPI and MPICH-G2 are presented in the accompanying report D-JRA2-1.3 [6].

3.1 Available Grids

3.1.1 Machine Specifications

The following machines were used in different contexts (i.e., standing alone or coupled as a cluster or grid) in the results presented in this work. The machines will generally be referred to throughout the report by the first part of their fully qualified domain name.

MACHINE	LOCATION	TYPE	PROCESSOR
e3500.epcc.ed.ac.uk	JCMB, Edinburgh	SUN e3500	400 MHz
lomond.epcc.ed.ac.uk	ACF, Edinburgh	SUN e15000	900 MHz
bluedwarf.egs.ed.ac.uk	ACF, Edinburgh	IBM p690	1.3 GHz
hpcx.ac.uk	Daresbury, England	IBM p690+	1.7 GHz
psi.rzg.mpg.de	Garching, Germany	IBM p690	1.3 GHz
zahir.idris.fr	Paris, France	IBM p690/p690+	1.3/1.7 GHz

Table 1. The names, locations and processor speeds of the different machines used as part of the study. Note that the physical distance between the ACF and the JCMB in Edinburgh is around 10km, while the distance between Paris and Garching is around 500km.

3.1.2 MPI Latency and Bandwidth

The latency and bandwidth for various configurations were measured using the same program performing repeated `MPI_Send()` and `MPI_Recv()` operations for messages of length between 0 and 1Mbyte. The benchmarks were not performed under special conditions, but are merely to illustrate the performance one might achieve during normal operation.

MACHINE/GRID	MPI	LATENCY	BANDWIDTH
e3500.epcc.ed.ac.uk	Native	6.7 μ s	230 MB/s
lomond.epcc.ed.ac.uk	Native	4.9 μ s	460 MB/s
bluedwarf.egs.ed.ac.uk	Native	8.7 μ s (150 μ s)	2.4 GB/s (210 MB/s)
hpcx.ac.uk	Native	7.5 μ s (90 μ s)	2.0 GB/s (330 MB/s)
psi.rzg.mpg.de	Native	10 μ s (130 μ s)	1.4 GB/s (120 MB/s)
zahir.idris.fr	Native	11 μ s (20 μ s)	1.3 GB/s (650 MB/s)
e3500 / lomond	PACX	2.4 ms	3 MB/s
lomond / bluedwarf	PACX	1.8 ms	10 MB/s
lomond / bluedwarf	MPICH-G2	660 μ s	11 MB/s
psi / zahir	PACX	16 ms	12 MB/s

Table 2. MPI latency and bandwidth recorded both intra-machine and inter-machine using different MPI implementations. Note that figures in brackets for native MPI on IBM machines use `MP_SHARED_MEMORY=no`, and so can be thought of as being indicative

of inter-node latencies and bandwidths on the p690 clusters which would come into play for any sufficiently large problem.

3.2 GADGET on the Grid

3.2.1 Homogeneous Cluster: SUN e3500 and e15000

The first grid available to the project was one based on two SUN machines at Edinburgh (the e3500 and lomond) using PACX MPI. This was used as the basis for a number of preliminary studies in which the following issues were investigated:

1. the impact of using PACX MPI instead of the native MPI within a machine (there should be none);
2. the performance for problems involving: gas only, halo particles only, and a mixture (one might hope the greater computational requirement for gas particles might decrease any impact of poor communication);
3. balancing load dynamically between machines working at different rates (based on measured time via the DEISA decomposition routines).

As the computational power of this grid was fairly limited, a relatively small problem size was chosen (264,144 gas or halo particles) and run for a small number of time steps (20). This means the distribution of particles remains essentially homogenous throughout the simulation.

Figure 5a shows the time per iteration as a function of the number of processors for a problem containing 262,144 halo, or dark matter, particles. A number of different results are presented. First, the left-hand panel shows the times for the problem run on each machine separately using either the native MPI or PACX MPI (the two-processor overhead of PACX MPI is ignored in these plots). The results show that the use of PACX MPI incurs no overhead compared to the native MPI; this should be expected as PACX MPI merely wraps the native MPI in this situation. In addition, the plot shows that the e15000 is somewhat faster per iteration than the e3500; again, this is to be expected from the machine specifications. Second, if one moves to a situation where GADGET is run using PACX MPI between the machines with an equal number of processors on each machine (right-hand panel in Figure 5a) the situation can be seen to be broadly the same. However, the performance is now somewhat degraded, with evidence of adverse scaling at the highest number of processors used. Further investigation showed that the increase in time was largely connected with increased time for `MPI_Alltoall()` associated with the FFT required for the particle mesh. In the right-hand panel, the difference between static and dynamic load balancing is also shown: if more work is allowed to migrate to the faster machine via dynamic load balance, there is a small increase in performance in most (but not all) cases.

Figure 5b shows the same results for a problem containing 262,144 gas particles. The increased time required for the SPH calculation is reflected in the difference in the vertical scales between 5b and 5a. Again, the left-hand panel demonstrates that there is little or no overhead associated with using PACX MPI per se compared with the native MPI. The inter-machine results (right-hand panel) again show broadly worse performance than the intra machine case even if dynamic load balancing is used.

Figure 5c shows results for a significantly larger problem: one which includes both 262,144 dark matter particles and 262,144 gas particles. The results are now more promising. The overall time taken for the inter-machine problem is the same as that taken for the intra-machine problem for the slower machine (when load balancing is static). This means that the performance in the inter-machine case is just limited by the slower of the two machines. If the load balancing is dynamic, then the inter-machine performance is half way between that of the two machines separately. This is the best that can be achieved in this case, and suggests that the increased work per process in this problem means that inter-machine communication is no longer limiting performance.

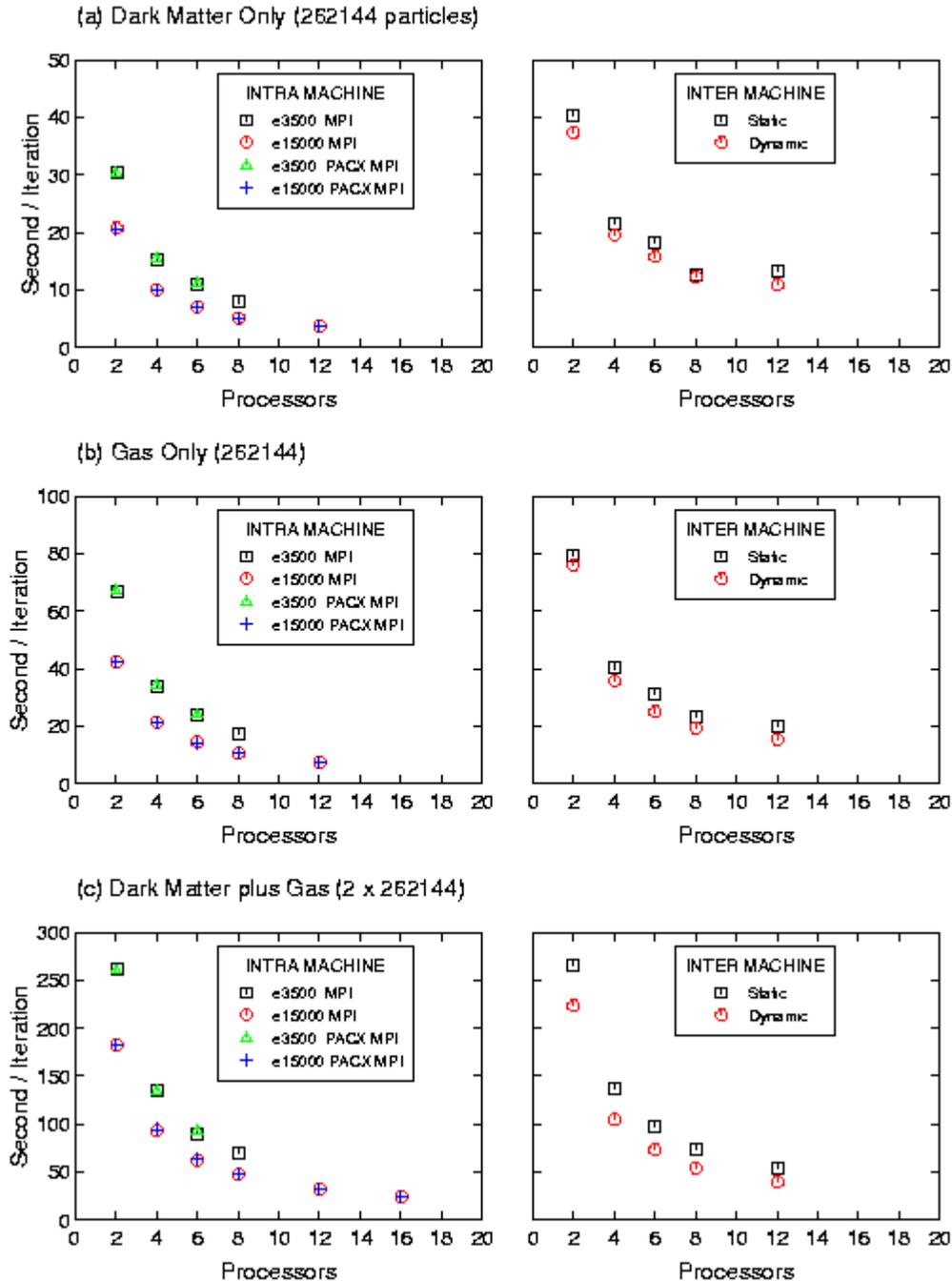


Figure 5. Intra- and inter-machine results using PACX MPI for a number of different problems (a) dark matter particles only, (b) gas particles only and (c) dark matter and gas particles combined. The left-hand panels show native and PACX MPI results within each machine, while the right-hand panels show the inter-machine results when load balancing is static and dynamic.

3.2.2 *Heterogeneous Cluster: SUN e15000 and IBM p690*

A second cluster at Edinburgh was used to investigate possible differences between the grid-enabled MPI implementations available.

However, the use of the heterogeneous cluster highlighted some potentially serious problems for GADGET. In particular, GADGET makes use of a number of complex data structures which consist of a mixture of basic data types (for example, 4 byte `ints` and 8 byte `doubles`). If such structures are aligned differently on different machines then passing MPI messages containing such structures as a single entity of type `MPI_BYTE` will not work. It has not been possible to control these alignment issues via compiler options in the case of the e15000 and the p690, so intervention in the code would be necessary to give a working version.

A solution to this problem would be to define MPI types for the message objects, which should then avoid alignment problems between platforms. This has not been pursued.

3.2.3 *DEISA Infrastructure*

A number of small test problems have been run on the DEISA infrastructure to ensure correct operation. These tests demonstrate that the operation of the meta-computer and PACX MPI were fundamentally sound. In order to assess the performance of an IDRIS-RZG meta-computer on a more realistic problem, a larger simulation of 256^3 particles was undertaken on 124 + 4 processors with PACX MPI (62 + 2 processors at each site).

The results of the exercise were not encouraging. The test failed to complete in the time allocated to it (being some 10 times the expected intra-machine time).

The problem occurred during the first episode of `domain_exchangeParticles()`, which is a potentially all-to-all operation implemented as a series of point-to-point communications. This will lead to message congestion in the PACX single in-server / out-server model with the relatively high number of processors. It is thus speculated that is the cost of the hopeless performance.

3.3 *General Comments*

The following general observations can be made about the results from the homogeneous SUN test grid:

- Clearly, inter-machine MPI latency and bandwidth appear poor compared with intra-machine values. This adversely changes the communication to computation ratio by a factor of roughly 500 in the case of the SUN cluster

- It is observed that iteration times of 0.1-1.0 seconds can support reasonable intra-machine scaling on the SUN machines. One would then expect a factor of 500 increase in computational effort would be required to rebalance the communication to computation ratio if a grid MPI were used. This is consistent with the results for the Sun cluster where at least 50 – 100 seconds per iteration are required to recover reasonable inter-machine performance.
- The PACX MPI overhead of two processors, while in itself acceptable, is annoying in practice as most HPC systems provide queues which have power-of-two numbers of processors. The overhead of two processors means one must then run the real problem with unfavourable numbers of working processors (particularly when using discrete Fourier Transforms).

Using these (rather simple) arguments one can make some statement about the DEISA Grid with knowledge of the measured MPI latencies:

- If one considers the IBM intra-node latency of 10 microseconds, the ratio of the intra-machine to inter-machine latency is around 1600. However, if one takes the inter-node latency of typically 100 microseconds, then the ratio is only 160. This suggests that a problem which exhibited reasonable scaling on the p690 cluster would have to be increased in size by about a factor of 200 to recover a viable communication to computation ratio on the DEISA grid.
- This suggests that if the problem of inter-machine message bottleneck could be addressed, massively parallel computations on a DEISA meta-computer could still be possible.
- Results suggest that special effort might be required in code to avoid heavy use of point-to-point mechanisms between machines. This means using optimised MPI functions such as `MPI_Alltoall()` and `MPI_Alltoallv()` wherever possible, which would minimise the number of inter-machine messages where implemented corrected.

4 Load Balancing

In this section we describe the domain decomposition used in the DEISA work and show that it incurs no additional overhead compared with the standard decomposition code in GADGET. Attempts at using measured time to balance load dynamically on a p690+ machine (HPCx) are also described.

4.1 The Domain Decomposition

4.1.1 Standard Decomposition Code

The standard (public release) GADGET2 code performs the domain decomposition in the following steps:

1. Each process builds a sorted list of particles in Peano-Hilbert order. This list is used to construct the local tree in terms of Peano-Hilbert segments.
2. The top level nodes of each local tree are then combined so that each process has a view of the global top level tree in terms of Peano-Hilbert segments.
3. The global number of Peano-Hilbert segments present to take part in the domain decomposition is counted (being $0 - N_{\text{Topleaves}} - 1$).
4. The number of particles of each type is counted in each Peano-Hilbert segment. The number of force calculations performed on particles in each segment *in the previous step* is counted as a measure of work.
5. The Peano-Hilbert segments are recursively decomposed in an initial attempt to balance work subject to the constraint that memory limits are observed.
6. The initial decomposition is refined in an iterative manner to optimise the load balance, again subject to memory constraints.

Note that steps 5—6 are repeated on each process, so all processes agree on the global decomposition.

4.1.2 DEISA Decomposition Code

In order that the decomposition be aware of the difference between clusters in a grid context, the code for the decomposition was rewritten. The opportunity was taken to combine steps 5 and 6 above into a single routine.

A problem with 16,777,216 (256^3) dark matter particles was run for a significant number of iterations to ensure the load balancing was tested for a non-homogenous distribution of particles. To compare the new decomposition routine with the original two pairs of runs were performed with `PartAllocFactor = 1.6` and with `PartAllocFactor = 3.2`. The results are shown in Figures 6 and 7, respectively. In the first case, the results

show that there is no overhead associated with the new code compared with the original when the load balancing is based on the number of interactions computed in each case. In the second case, the new code remains competitive.

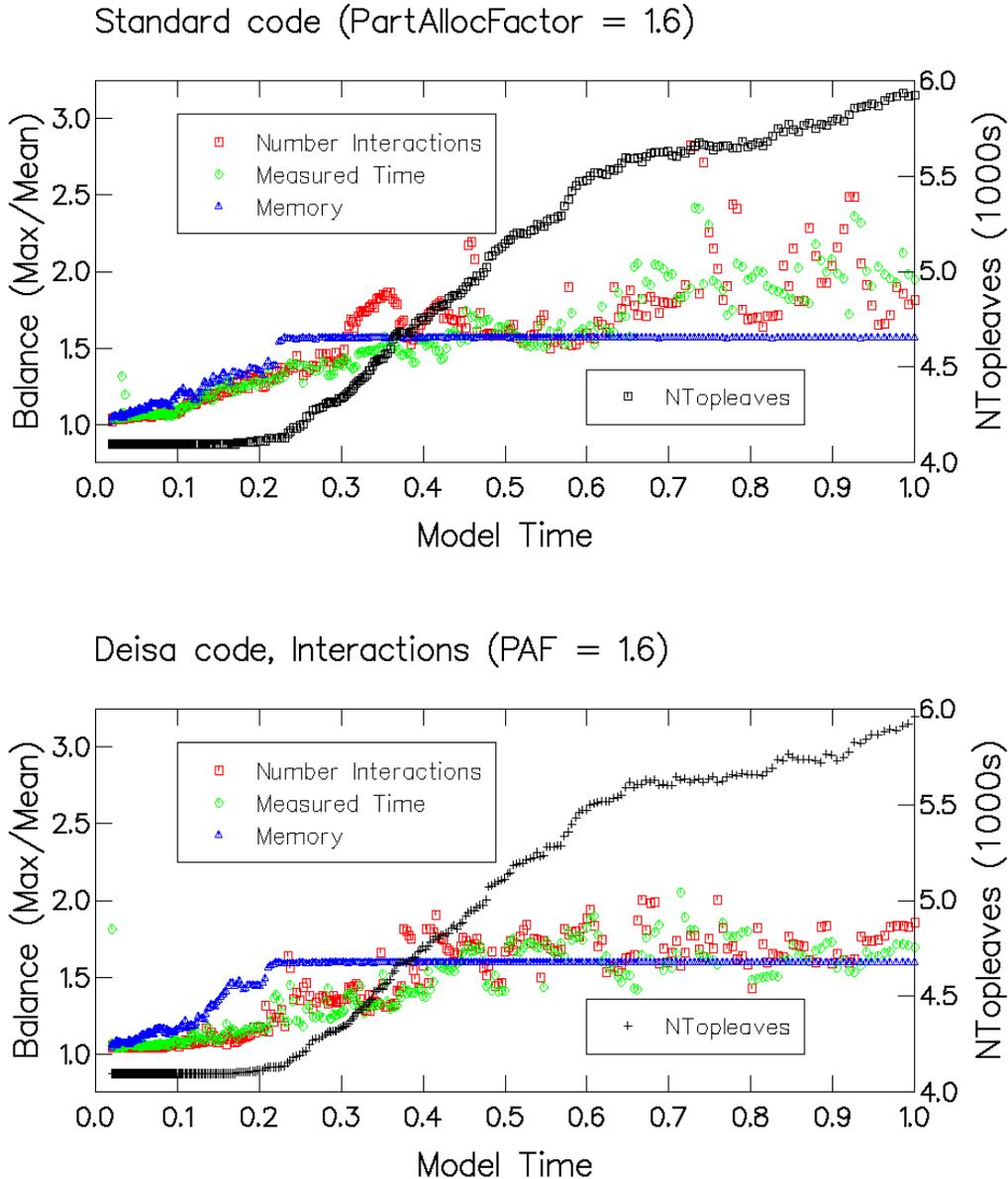


Figure 6. Load balance in terms of the number of interactions computed, the measured CPU time, and the memory as a function of model time. The load balance is expressed as the maximum divided by the mean over the current number of `Topleaves` (also plotted on the right-hand scale). Load balancing for the DEISA code is based on the number of interactions computed, as is load balancing in the original code.

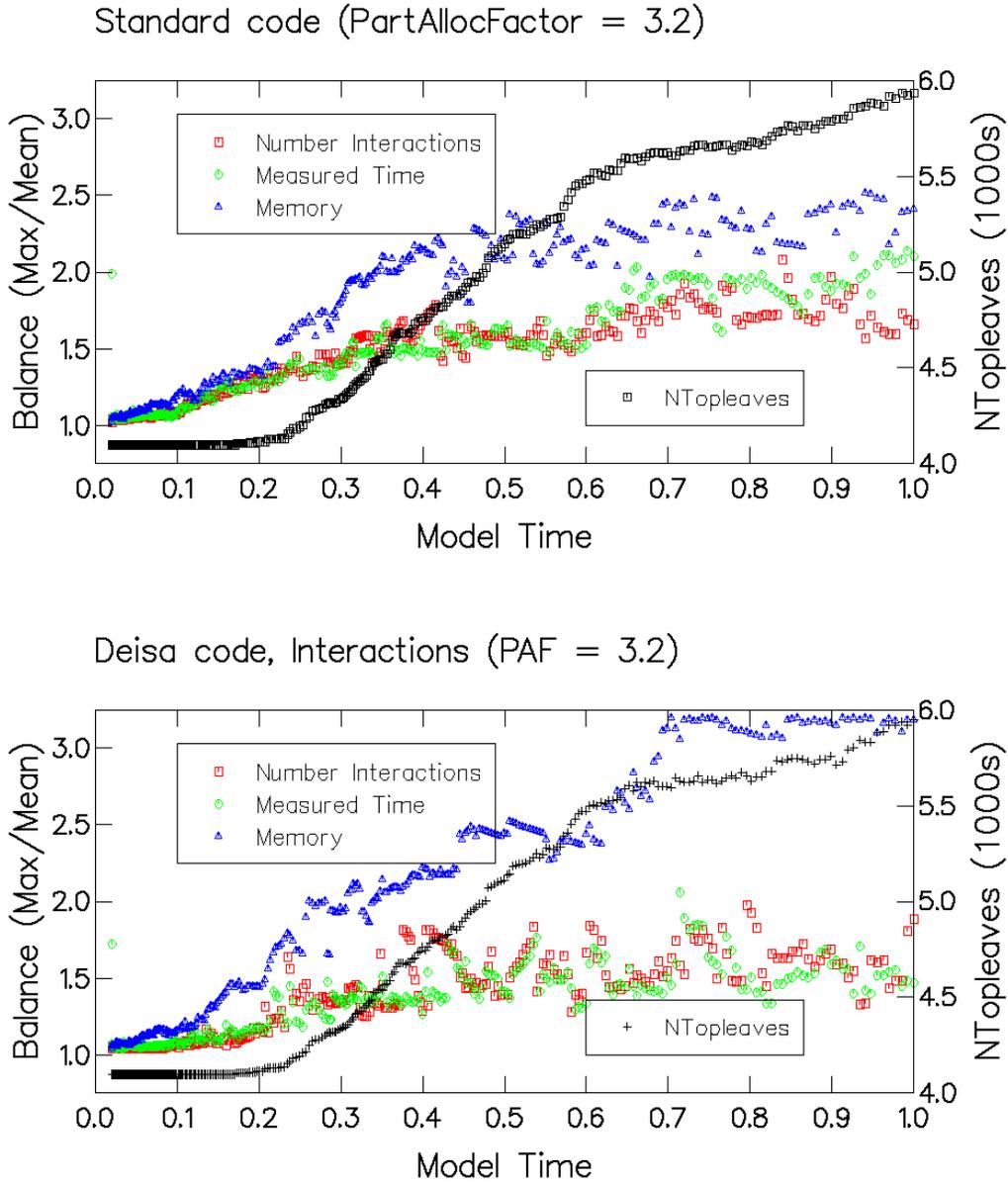


Figure 7. As for Figure 6 but with `PartAllocFactor = 3.2`.

4.2 Dynamic Load Balancing

The DEISA decomposition code was further adapted to allow a decomposition based on the measured CPU times for certain instrumented parts of the calculation. This would allow load to be balanced when machines of different speed were including in a meta-computer (as in the SUN test grid). In this situation, the CPU time might be a more reliable measure of load than the number of interactions computed. (For dark matter particles this means the force calculation, but can be extended to include the SPH

calculation for gas particles.) This measured time is purely a CPU time, and does not include any effect of communication or synchronisation, and so should be representative of the real computational load.

Results for the same problem as considered in the previous section are shown in Figure 8. It can be seen clearly that the introduction of the measured time has a strong negative effect on load balancing. In fact, a strong oscillation can be seen which means the decomposition can vary wildly between iterations, a fact that further degrades performance owing to large numbers of particle movements between processors. No viable solution to this instability was identified.

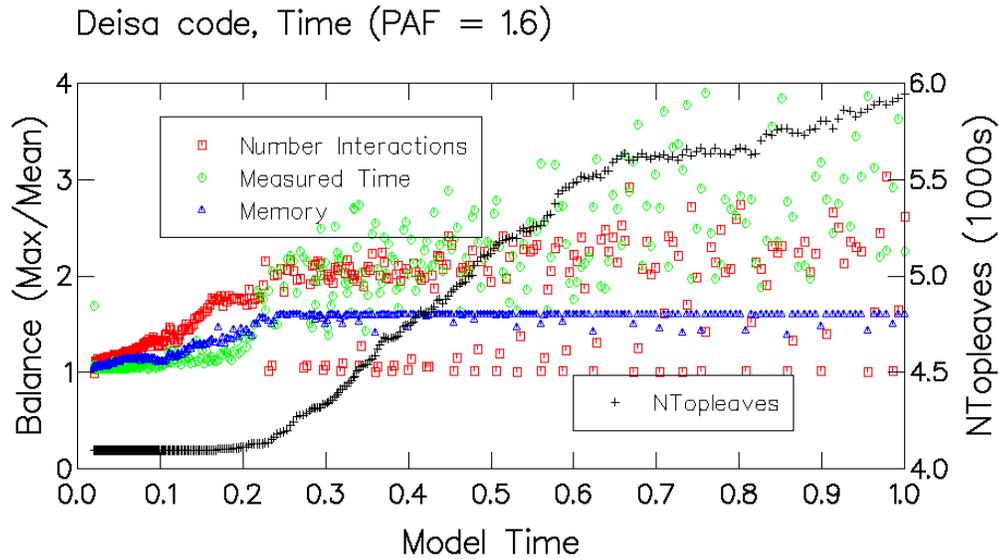


Figure 8. Load balance by the DEISA code when measured time is used in determining the decomposition.

5 Conclusions

The GADGET code has been adapted to run using PACX MPI or MPICH-G2, and hence on a meta-computer or Grid.

Initial results for relatively small problems run on up to 12 processors suggested that the relatively poor inter-machine MPI latencies and bandwidths could be overcome for a large enough problem. However, tests on a larger scale (128 processors) on the DEISA infrastructure proved unsuccessful.

Based on simple measurements of the current MPI latency and bandwidth for the DEISA network, one could speculate about the size of problem which would be required to balance the computation to communication ratio. This would be comparable to the largest runs performed with the code as it currently stands. Simulations on this scale would bring with them additional questions about network reliability and so on, which would need to be answered to ensure success.

However, with current technology, large Grid-enabled computations for GADGET do not appear to be viable.

The main results are as follows. The grid-enabled version of GADGET is as good as the original form, which allows the original objective to be achieved if the underlying infrastructure is suitable. The performance of the grid-enabled code on DEISA is below expectation, due to bandwidth and latency. The new load balancing strategy does not improved performance.

In conclusion, to use the grid-enabled version of GADGET2 in a metacomputing environment, one requires an extreme problem size and the ability to co-schedule tasks.

Finally, for the near future, the present DEISA strategy to run the complete application on the largest available system is sound.