University Politehnica of Bucharest
Faculty of Automatic Control and Computers
Computer Science and Engineering Department

Diploma Thesis

N-Body Simulations With GADGET-2

by

Victor-Lucian Spiridon

Supervisors:
Associate Prof. Dr. Eng. Emil Sluşanschi
Dr. Marian Şuran – Astronomical Institute of the Romanian Academy

Bucharest, July 2011

Contents

List of Figures

List of Tables

Abstract

In this thesis we choose to discuss cosmological N-Body/SPH simulations on massively parallel computers with distributed memory using GADGET-2. Afterwards we target the porting of GADGET-2 on the NCIT (National Centre for Information Technology) computer cluster at Politehnica University of Bucharest with an analysis of the scalability for several simulations.

GADGET-2(**GA**laxies with **D**ark matter and **G**as int**E**rac**T**) is a novel cosmological simulation code, publicly available, developed by Volker Springel at Max-Planck-Institute for Astrophysics in Munchen, Germany as an improved version of GADGET. It is a massively parallel code that uses an explicit communication model implemented with the standardized MPI comunication interface. It can run on systems ranging from individual PCs to cluster of workstations and supercomputers.

We also describe in a briefly manner the methods and algorithms implemented in GADGET-2 and implied within different types of simulations. They are important because they help understand the inner-workings of the code. Also we describe the terms used widely in the astrophysics comunity such as: Redshift, Periodic boundary conditions, Periodic box, Comoving coordinates, Isotropy.

This thesis can be used as a manual for running GADGET-2 simulations on the NCIT cluster, containing information about every step in the process: instalation, tuning, generating initial conditions, configuration, running, post-processing, data visualization and analysis. All software packages used are freely available.

The scalability analysis is done on different types of simulations ranging from a few thousand particles to 768^3(452.984.832) particles. The size of the simulation is limited by the amount of available physical memory(RAM) when provided with sufficient computing time. A trend is established stating the computational resources necesary with respect to the size of the simulation.

Chapter 1

Introduction

This thesis is about "scientific computing" applied to Astrophysics in the form of cosmological computer simulations. But first of all, what is a computer simulation? Generally speaking, a simulation is an imitation of a real thing or process that generally entails representing certain key characteristics or behaviours of a selected physical or abstract system. When the imitation is realized with the help of computers we talk about a computer simulation. The phenomena can vary from simple problems such as the sliding of an object on an inclined plane to very complex problems such as weather forecast, aerodynamics, material deformation and molecular processes.

Not long ago, scientists were making research based on a "cut-and-try" approach. To give an example let's consider the aerodynamics of a race car. The designers had to make scale models in order to precisely determine the performance of the design. If the design wasn't good enough, the making of a new one could take a very long time, thus limiting the productivity. This leads to compromises that translates in lower performance. Nowadays, efficient scientists have a "simulate-and-analyze" approach. Aided by computers, the designers can test and create a greater number of designs, because it takes less time to obtain the performance parameters.

The simulation as a scientific approach can be viewed mainly as composed of two steps. First, the physical system is described in a form governed by a set of equations, which represent continuum approximations. Such approximations are not possible for all systems. Second, the experiment in the laboratory is replaced by the simulation of the model described by the equations, i.e., a numerical experiment based on a discrete model. The data output of the simulation often represents states of the system at specific moments in time. This can be later interpolated to obtain a continous visualization of the results. In order to display the results, the data must pe post-processed which is not a trivial process. In fact

it can pose serious handling problems because the output files can be very large: A snapshot of the system of 768^3 particles, detailed in a later chapter, has 12 Gigabytes in size, and a video of the process from beginning to end requieres at least 100 snapshots. Working with large datasets raises all sorts of new problems such as reading and writing in parallel on a distributed file system, requiring additional programming overhead.

After showing in a short manner what a computer simulation implies, we observe that the "simulation scientist" is at the intersection of three large domains. He must be well prepared in topics such as: computer science (hardware and software programming), mathematics and physics, numerical methods and physical modelling as depicted in Figure 1.1 [Mil09].

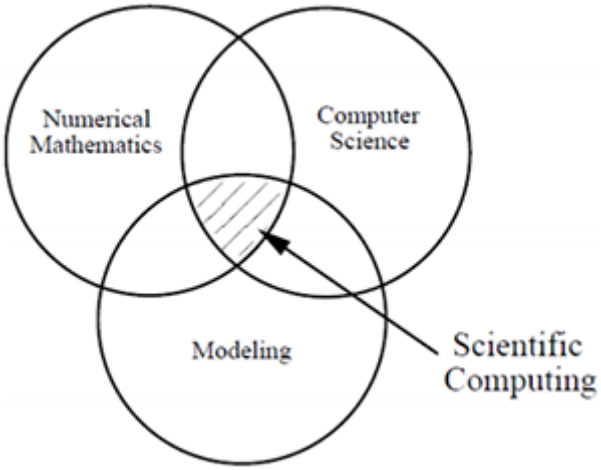

Figure 1.1: Scientific Computing model.

An aerodynamic computer simulator is built around a proven model. The results can be verified with real and exact observation in order to validate the correctness of the algorithm. This cannot be said about cosmological N-Body/SPH computer simulators and that is because they are built around an unproven model about the universe. Their purpose is not necesarily to take a glimpse into the future, but to trace the history and the evolution of the universe. That means a cosmological structure formation simulation usually runs from a given time in the past until present. Then the results are compared with observation in order to validate the model which is our motivation. Only with a valid model a simulation scientist can "predict" the future, ultimately the final goal.

GADGET-2 is built around the ΛCDM model, today's leading theoretical paradigm for structure formation which states that the universe is made out of 70% dark energy, 25% dark matter, and 5% baryon matter (real matter), see Figure 1.2 [Inc11e], expanding with a growing rate. The aim is to give astro-

physicists a research tool in order to prove the correctness of this model and further improve it.

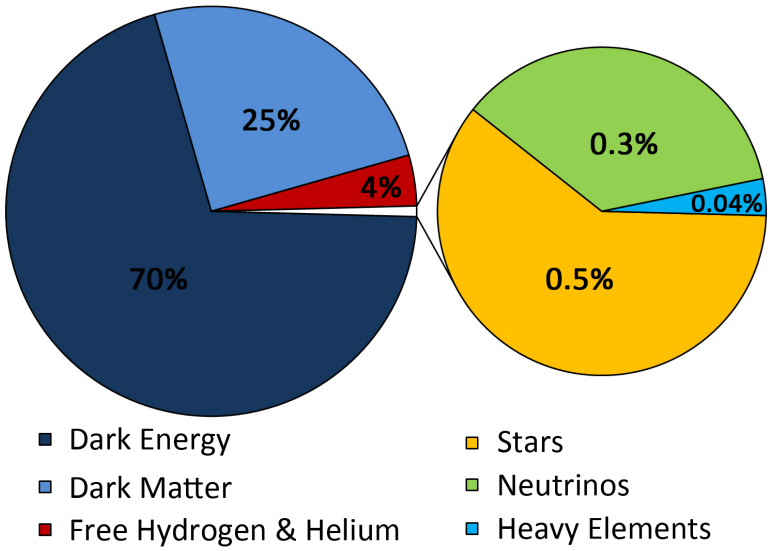

Figure 1.2: The proportional composition of the Universe.

This thesis purpose is to introduce the reader to cosmological simulations and focuses on the steps of using GADGET-2 on the NCIT Computer Cluster.

• Chapter 2 – "Problem Statement" will define the N-Body Problem, and at the end it will present some related work in N-Body/SPH simulations.

• Chapter 3 – "Algorithms and Concepts Behind GADGET-2" is the chapter were there are presented some important algorithms and ideas necessary in order to understand at a basic level the nature of cosmological structure formation simulations, as well as the inner workings of the code.

• Chapter 4 – "Hardware and Software Configuration" presents the NCIT Computer Cluster resources, as well as the steps necessary to install and run simulations with GADGET-2.

• Chapter 5 – "Simulations with GADGET-2" explains the basics of building initial conditions (snapshots of the system used as a starting point of the simulation) and runs through the major types of simulations.

• Chapter 6 – "Case Studies" is the chapter that contains the results, the peformance indicators and profiling for 4 different simulations.

• Chapter 7 – "Conclusions and Future Work" presents the conclusions of the simulations and establishes a trend in performance for cosmological simulations. It also contains ideas that can be implemented in the future in order to create a powerful platform for Cosmological N-Body simulations with GADGET-2.

Chapter 2

Problem Statement

The N-Body problem is at the heart of simulations in the modern astrophysical domain. With this thesis are searching for a "state of the art" tool that employs modern techniques such as Particle Mesh and Smoothed Particle Hydrodynamics in solving N-Body problems. In particular, this tool must handle large numbers of particles in the order of hundreds of millions in an efficient and scalable manner. But first of all, what is the N-Body problem?

2.1 Problem Exposure

The general N-body problem is usually formulated for $N \geq 2$ as follows. In cgs (centimeter - gram - second) system of units and for $i = 1, 2, \ldots, N$, let P_i of mass m_i be at $\vec{r}_i = (x_i, y_i, z_i)$, have velocity $\vec{v}_i = (v_{i,x}, v_{i,y}, v_{i,z})$, and acceleration $\vec{a}_i = (a_{i,x}, a_{i,y}, a_{i,z})$ at time $t \geq 0$. Let the positive distance between P_i and P_j, $i \neq j$, be $r_{ij} = r_{ji} \neq 0$. Let the force on P_i due to P_j be $\vec{F}_{ij} = \vec{F}_{ij}(r_{ij})$, so that the force depends only on the *distance* between P_i and P_j. Also, assume that the force $\vec{F}_{ji}$ on P_j due to P_i satisfies $\vec{F}_{ji} = -\vec{F}_{ij}$. Then, given the initial positions and velocities of all the P_i, $i = 1, 2, \ldots, N$, the general N-body problem is to determine the motion of the system if each P_i interacts with all or part of the other P_j's in the system [Mil09].

$$N = \begin{cases} 2 \leq N \leq 10^2 & \text{solar system} \\ 10^2 \leq N \leq 10^6 & \text{galaxy, cluster of galaxies} \\ 10^5 \leq N & \text{cosmological structure formation} \end{cases} \tag{2.1}$$

Solving the N-Body means determining the acceleration, velocity and position of the N particles out of Newton's equations of motion. Interactions between the particles (governed by the gravitational force) form a system of N differential equations. The problem is categorized according to the choices in (2.1).

2.2 Related Work

Since the first simulations in the 1970s, there is a constant effort of producing powerful N-Body codes for cosmological simulations. The first simulations evaluated the forces on particles by direct summation(PP) of the Newtonian interaction between particle pairs, a very inefficient method when the particle number exceeds 1000. Tree codes radically reduce the cost by grouping distant particles into aggregates, an then summing over such aggregates rather than over individual particles. PM codes estimate the density on a grid an then use DFTs to convolve the density with the Green's function. A short review about various techniques employed in N-Body codes can be found in [Kne01].

GADGET-2 [Spr05a] is a modern TreePM code which has increased performance over the plain PP method and a SPH technique for simulating gas. G2X [Fri09], is a hardware accelerated version of GADGET-2 using the CUDA technology on GPGPUs that has a speedup of 10 and even 20 over the original version. However, it is highly limited by the GPU's memory to simulations in the order of 10^6 particles.

A hybrid between the PP method and the PM method is the P^3M method with which the short-range force is computed with the PP method and the long-range force with the PM method. The adaptive $P^3M(AP^3M)$ code, used in AMIGA [Kne01] and ART [Kra99], computes high density regions by an additional P^3M calculation in which a finer grid covers the region, avoiding the increase of the number of particles computed with PP, othwerwise the calculation becomes prohibitively costly. Advantages of this method are that they naturally admit periodic boudary conditions, adaptive softening and individual time-steps. However this method doesn't easily admit SPH techniques for very clustered regions which require exquisite spatial resolution.

RAMSES [Tey02] is a newer code that uses an N-Body solver very similar to the one developed for the ART code. However it also includes a hydrodynamical solver based on a second-order Godunov method, a modern shock-capturing scheme known to compute accurately the thermal history of the fluid component.

A very modern code is VINE [Wet09] written in Fortran95 using a modular structure that features, multiple integrators, individual time-steps, SPH, free or periodic boundaries and support for GRAPE hardware [NBo]. Beside the N-Body and the SPH particles, a third particle species can be included to model massive point particles which may accrete nearby SPH or N-body particles(e.g., stars in a molecular cloud). It is faster than GADGET-2 by a 2 to 5 factor, depending on the simulation type, but since it's parallelized with OpenMP, the number of processors used and the size of the simulations are limited.

Amongst the large number of N-Body codes, GADGET-2 is the most famous one, thanks to the "Millenium Run" simulation at the Max Planck Society's Supercomputing Centre in Garching, made public by a press release in June

2005 [Max05]. This wouldn't have been possible if it wasn't for GADGET-2's high scalability, the "Millenium Run" being the largest N-Body simulation carried out so far, even until this day, with more than 10^{10} particles. The results of the simulation match observed cosmic structures such as the Sloan Great Wall, the CfA2 Great Wall, and the 2dFGRS(galaxy redshift survey) - Figure 2.1:

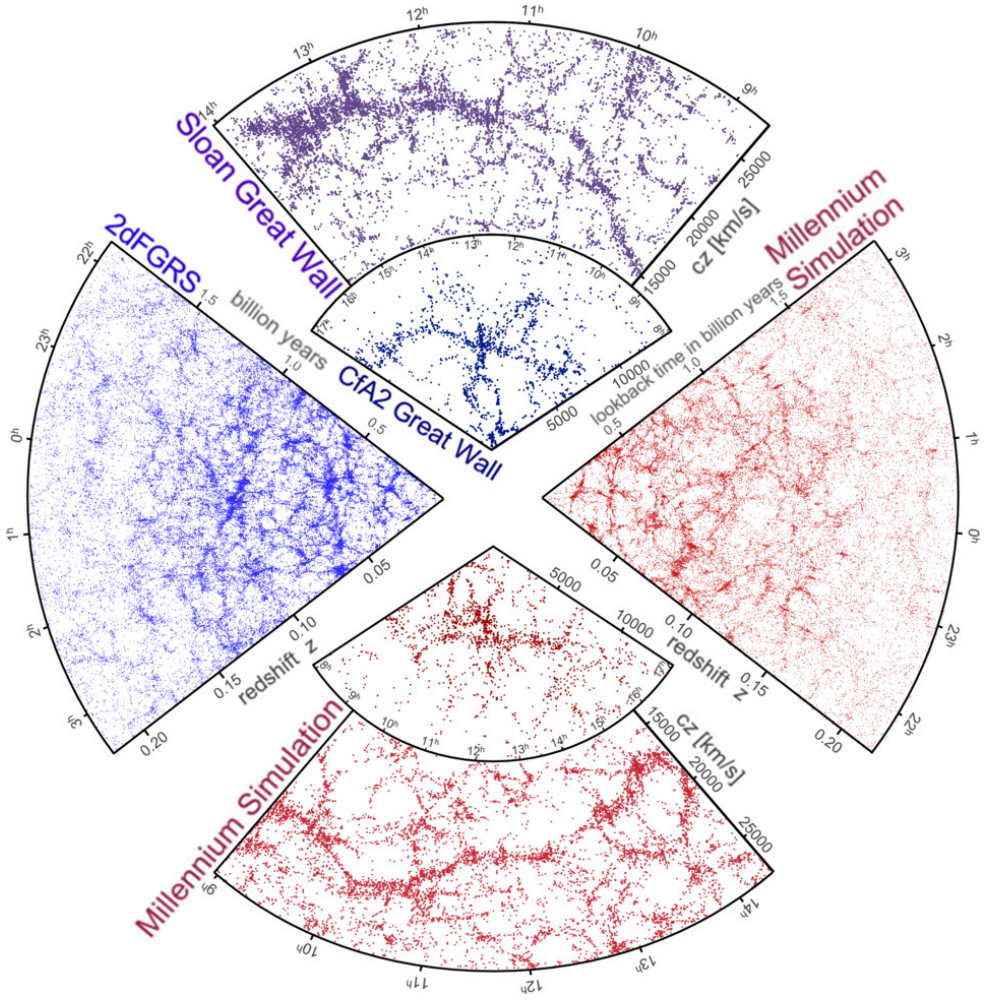

Figure 2.1: Walls and cosmic structures comparison between astronomical observations(blue and purple) and the Millenium Run(red).

Taking this results into account, the codes scalability and the large number of publications that have directly used the Millenium Run simulation data [Lem], we decided to use GADGET-2 as the code for running N-Body simulations on the NCIT Cluster, the main topic of this thesis.

Chapter 3

Algorithms and Concepts Behind GADGET-2

3.1 Basic N-Body

3.1.1 The General N-Body problem

Solving an N-Body problem means determining the velocities and the positions of the particles at different moments in time. In writing the equations needed for solving an N-Body problem, we can start from Newton's Universal Law of Gravitation stating the attraction force between 2 particles:

$$\vec{F}_{ij} = G\frac{m_i m_j}{r_{ij}^3}\vec{r}_{ij} \tag{3.1}$$

where:

- $\vec{F}_{ij}$ is the force between particles i and j
- G is the universal attraction constant
- m_i and m_j are the particles masses
- $\vec{r}_{ij}$ is the distance vector between the particles

For a system of N particles the attraction force for particle i is:

$$\vec{F}_i = Gm_i \sum_{\substack{j=1 \\ j\neq i}} \frac{m_j}{r_{ij}^3}\vec{r}_{ij} \tag{3.2}$$

Newton's Second Law for particle i is:

$$\vec{F}_i = m_i\vec{a}_i \tag{3.3}$$

When we add (3.2) in (3.3) we obtain the following formula for the acceleration of particle i:

$$\vec{a}_i = G \sum_{\substack{j=1 \\ j \neq i}} \frac{m_j}{r_{ij}^3} \vec{r}_{ij} \tag{3.4}$$

Then the problem is formulated as an ordinary differential equations system, which equations are derived from *Newton's equations of motion*:

$$\begin{cases} \dfrac{d}{dt}\vec{r}_i = \vec{v}_i \\ m_i \dfrac{d}{dt}\vec{v}_i = \vec{F}_i \end{cases} \tag{3.5}$$

inserting one equation into another in (3.5) and (3.4) in (3.5) we obtain the final system:

$$\begin{cases} \dfrac{d^2}{dt^2}\vec{r}_i = \dfrac{\vec{F}_i}{m_i} \\ \dfrac{d}{dt}\vec{v}_i = \dfrac{\vec{F}_i}{m_i} \\ \vec{a}_i = \dfrac{\vec{F}_i}{m_i} \end{cases} \tag{3.6}$$

with $\vec{F}_i$ in (3.2).

Having this system of differential equations it's not hard to see that if we integrate the first and second equations in (3.6) we can determine the velocity and the position of the particle.

3.1.2 Solving the problem

To solve the (3.6) differential system we use numeric integrators such as Euler, Leapfrog, Runge-Kutta, Hermite, etc. An integrator integrates an equation only at specific moments in time, obtaining it's value. The period between the moments in time is called a timestep. The smaller the timestep, the smaller the integration error, and also the greater the order of the integrator the smaller the error. The order of an integrator is given by the exponent of 10 that divides the error. For example: in the case of a 2^{nd} order integrator, making the *timestep* 10 times smaller will make the *integration error* 100 times smaller. Also, in general, the higher the order of an integrator the more computational time it needs and the harder it is to program it.

As a rule, we usually need a more precise integration when the velocity of at least one particle is large, and usually the closer two particles are the greater their velocity. This is because the particle travels a bigger distance in that timestep, increasing the chance for errors from *events* not taken into account, ocurring during the timestep. Errors can lead to very unrealistic simulations in high kinetic energy configurations of particles such as binaries, see [Inc11a], which are

used mainly to model collisional systems. Making the timestep smaller means the simulation will run more slowly. Since we usually don't need small timesteps for all the particles, we must employ methods with individual timesteps, that are more difficult to code. Below, the plot of an orbit computed with a first order Forward Euler integrator, see Figure 3.1 [Mil09].

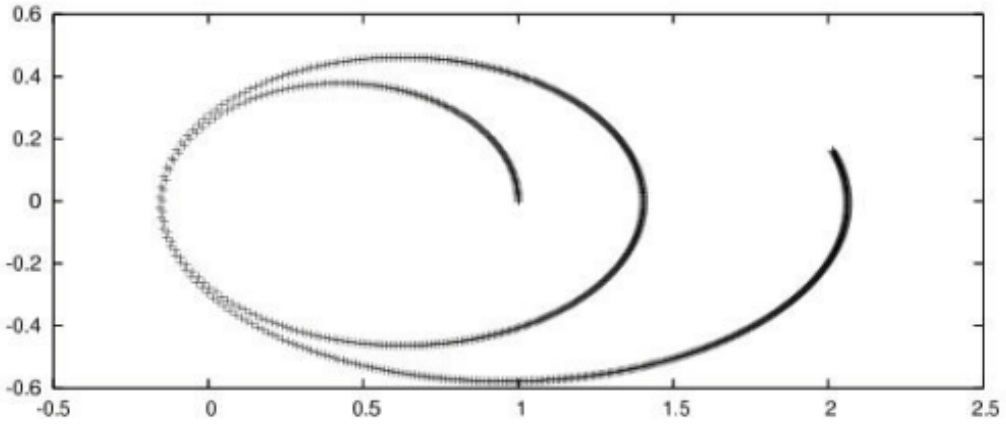

Figure 3.1: Forward Euler orbit altering caused by a not small enough timestep.

Choosing between integrators depends on the type of N-Body simulation we choose from the choices listed in (2.1). For a small N, such as a solar system, best suited is a high order integrator (12^{th} order) because it minimizes the errors. However, high order integrators not always accept variable individual timesteps, making them very good only for well-behaved systems (without particles that get very close).

For medium sized N integrators of 2^{nd} to 4^{th} order, with indivitual timesteps, are best suited. Because N is fairly large, a classic N-Body code will run very slow because of it's high complexity of $O(N^2)$. In this case more complex algorithms are used such as the oct-Tree algorithm which has a complexity of O(Nlog(N)). Because N is so large the system can't be simulated exactly but in an aproximate way, and because of the size and the nature of the simulation, it becomes realistic.

For N in the order of millions it becomes a problem of structure. The simulation doesn't track individual particles anymore, instead it targets the clusters, their shape and interactions between them. Interactions of particles within the cluster are not necessarly accurate, but at a macro scale the errors tend to cancel each other.

3.1.3 Gravitational Softening

Because of the very large dynamic range of the force, if two particles are too close, the force can take a very large value, close to infinity (in 32-bit floating

point representation of real numbers and even in 64-bit). A very large force means a very small timestep that tends to 0, which makes the simulation stop. Therefore, a method called *gravitational softening* is employed in order to prevent particles getting too close to each other. The larger the gravitational softening factor the smaller the *collisionality* of the simulation, to the extent that this kind of simulations are plainly called *collisionless*.

The method consists in altering the (3.4) formula to:

$$\vec{a}_i = G \sum_{\substack{j=1 \\ j \neq i}} \frac{m_j}{(r_{ij} + \epsilon)^3} \vec{r}_{ij} \tag{3.7}$$

where ϵ is called *softening length*. ϵ must be small when compared to the dimensions of a galaxy.

For more information about N-Body simulations in general see [Sch07]. In the next sections we will discuss some important algorithms and concepts implemented in GADGET-2 relevant to the types of simulations and basic run of the code.

3.2 The Tree - Particle Mesh Algorithm and Parallelization

GADGET-2 is designed for large numbers of particles and uses a Tree - Particle Mesh algorithm (TreePM) which has a complexity of O(Nlog(N)). The gain in speed is obtained by a small degree of approximation of the force of distant particles. In the following we will present briefly this algorithm. For more details, see: [BH86], [SYW00], [Spr05a].

3.2.1 Tree Algorithm

When we look at night at the sky, we only see points of light. We might think that each point represents a very powerful or a very close star, but when we look closer (using very powerful telescopes) we see that some points are in fact binary stars, some are solar systems much like our own, and some are entire galaxies. The galaxies are so far away that they appear only as points on the earth's night sky. This analogy can be made also for the gravitational force meaning that a galaxy far away can be aproximated to one single particle of mass equal to the entire mass of the galaxy located at the center of mass of that galaxy, see Figure 3.2. Barnes and Hut [BH86] introduced in 1986 such a method for solving N-Body problems that require less computation that the direct method. This method is called Barnes-Hut algorithm, tree-code, or hierarchical tree-code. The tree is called bh-tree or oct-tree.

Building the tree is fairly intuitive. The root of the tree contains all bodies in the simulated system (the entire volume). The tree is constructed from the

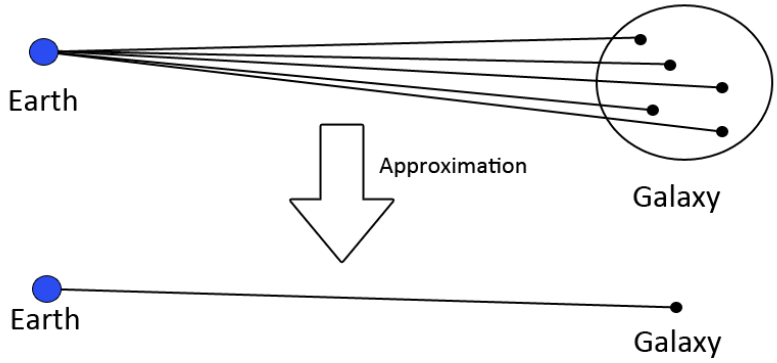

Figure 3.2: Gravitational potential of the galaxy approximated as the potential of a single particle.

root by splitting the volume associated with the node into 8 equal cubes (the length of the edge of the cube split in half). Cubes that contain no particles are ignored, and cubes that contain only one particle are called *leaves* (they don't have descendants.) The cubes that contain more than 1 particle are called *nodes*. Each node has associated with it the center of mass of the set of particles

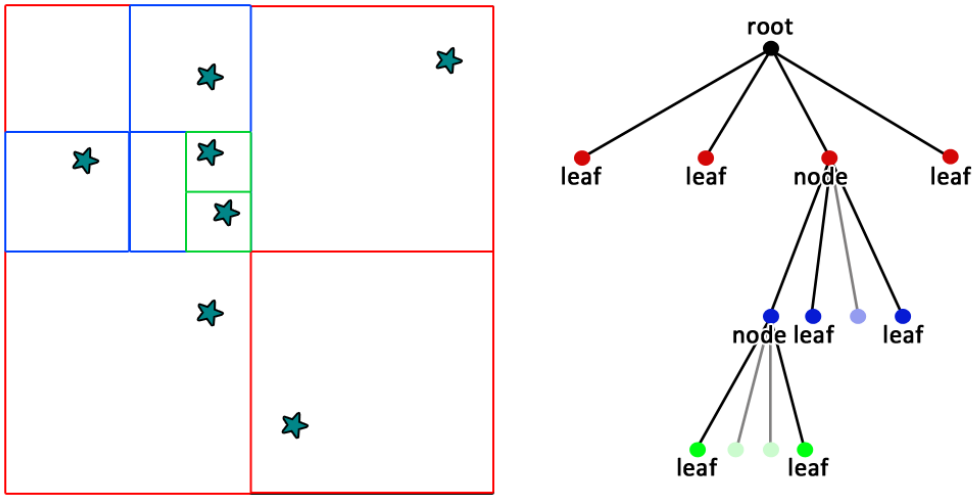

Figure 3.3: 2-dimensional system of particles with its corresponding bh-tree.

it contains and their total mass(this calculation is performed in one pass from leaves to the root). Each node is split until only leaves remain. The tree is

rebuilt for every timestep. In Figure 3.3 is an example of a particle system and its corresponding bh-tree in two dimensions, for simplification.

3.2.2 Parallelization

In the GADGET-2 implementation of the tree-code the simulated volume is divided by a space-filling fractal, a Peano-Hilbert curve. The local depth of the curve is regulated by the local particle density and work-load, such that the decomposition becomes naturally finer in high-density regions. Example of a Peano-Hilbert curve in two and three dimensions below, Figure 3.4.

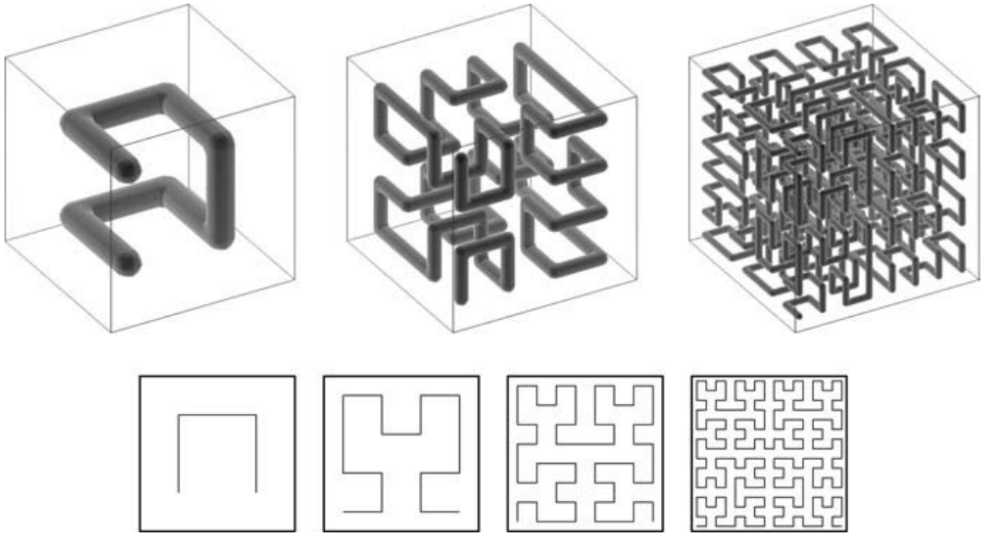

Figure 3.4: Space-filling Peano-Hilbert curve in two(bottom) and three(top) dimensions.

Then the 1 dimensional curve, containing all the particles, is split among the processors, inducing a domain decomposition in *blocks* of approximately the same computational work-load. The advantages of the Peano-Hilbert curve are that it allows for domains of arbitrary shape, while at the same time it always generates a relatively small surface to volume ratio for the domains, and this compactness reduces communication overheads.

3.2.3 Force Computation

For each particle the tree is *walked* from the root down, and if the node is sufficiently distant from the selected particle, then the node is added to the interaction list. If the distance between the selected particle and the node (its center of mass) is not sufficient then the node is *opened* and a distance check is performed between the particle and the descendants of the node. When is the node *sufficiently*

distant? In 1986 Barnes and Hut introduced an opening criterion stating that a particle is *sufficiently distant* when:

$$d > \frac{l}{\theta} \tag{3.8}$$

where:

- d is the distance between the selected body and the mass center of the node
- l is the length of the cube's edge associated with the node
- θ is the opening angle

Parameter θ determines the speed-up of the algorithm and its accuracy at the same time. It is suggested to choose this parameter in the range $0.7 \leq \theta \leq 1$. Smaller values of θ lead to the opening of a larger number of nodes, to a better accuracy of force computation and to computational slow-down. Usually $\theta = 1$ gives accelerations with errors around 1%. For a simulation with 10^6 particles, typically there are ∼1000 interactions per particle on average, making the algorithm significantly faster than a direct summation method [Dub96].

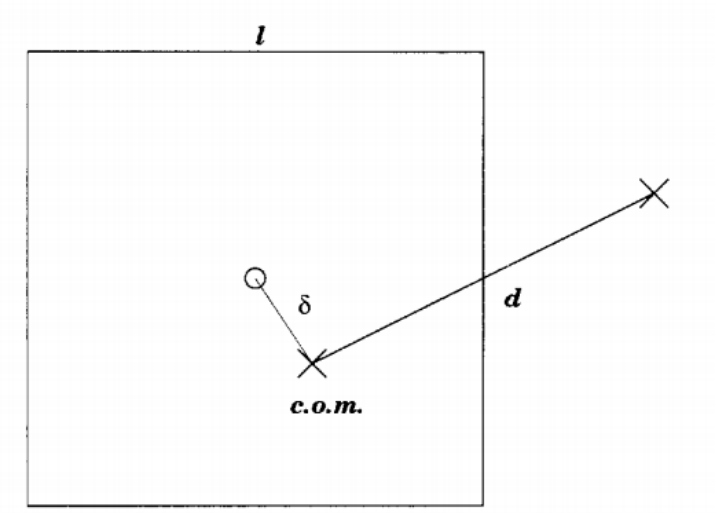

Figure 3.5: Geometry of the Barnes new opening criterion. The old Barnes-Hut opening criterion ignores δ.

However, this criterion can cause gross errors in some cases in which the center of mass is near de edge of the cell, and in 1994 Barnes introduced a new criterion:

$$d > \frac{l}{\theta} + \delta \tag{3.9}$$

where δ is the distance between the cell's center of mass and its geometric center.

3.2.4 Particle Mesh

GADGET-2 optionally allows to replace the pure Tree algorithm with a hybrid method consisting of a synthesis of the Particle Mesh(PM) method and the tree algorithm. The short-range part is then computed with the tree and the long-range part is computed with PM methods, thus maintaining the speed bonus of the tree algorithm. See in Figure 3.6 the split between the components of the gravitational potential [Bag02].

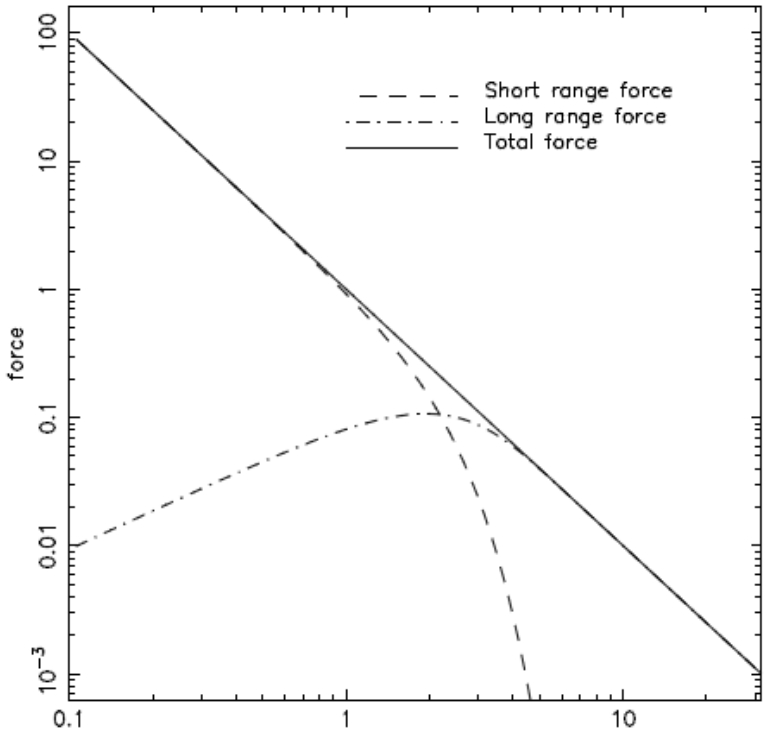

Figure 3.6: The split between the short-range component and the long-range component of the gravitational potential in the TreePM algorithm.

The principle of this method is that the system of particles is converted into a mesh of density values. The gravitational potential is then solved for this density grid, and using it the long-range component is computed using fourier based techniques. GADGET-2 uses a clouds-in-cells(CIC) assignment method to construct the mass density field on to the mesh [Spr05a]. This method is very usefull for simulations in periodic boxes because the tree only needs to be walked in a small spatial region aroung each target particle, and no corrections for the periodic boundary conditions are requiered. In addition, typically there is a gain in accuracy in the long-range force, which is now basically exact, and not an approximation as in the pure tree method.

Periodic boxes will be discussed briefly in a later chapter.

3.3 Smoothed-Particle Hydrodynamics

Smoothed-Particle Hydrodynamics(SPH) is a computation method used for simulating fluid flows. GADGET-2 uses this method in order to simulate its *gas* particles. You can imagine that physically speaking giant clouds of cosmic dust are very different from planets. Furthermore, they are modeled differently. The normal particles are zero in size (points in space), with the mass concentrated in that point. Their density is therefore infinite so they act pretty much like little black wholes. The SPH particles represent a volume of fluid with a specific density, and they can interact with each other not only by means of gravity but also by means of hydrodynamical flows.

The SPH particles is a discretization of a continuum phenomena such as a fluid. These particles have a spatial distance(known as the *smoothing length*), over which their properties are *smoothed* by a kernel function. In order to obtain the physical value of a property(such as temperature) in one place in space, one must sum up the contributions of neighbouring particles that are within the smoothing length. The value is given by the kernel function which is a function of distance. The smoothing length can be adapted to the environment sorrounding the particle, increasing it if the region has a small density or reducing it for clustered regions, so increasing the resolution.

The kernel function W used by GADGET-2 is:

$$W(r,h) = \frac{8}{\pi h^3} \begin{cases} 1 - 6\left(\frac{r}{h}\right)^2 + 6\left(\frac{r}{h}\right)^3, & 0 \leq \frac{r}{h} \leq \frac{1}{2} \\ 2\left(1 - \frac{r}{h}\right)^3, & \frac{1}{2} \leq \frac{r}{h} \leq 1 \\ 0, & \frac{r}{h} > 1 \end{cases} \tag{3.10}$$

where:

- r is the distance between the location in space and the SPH particle
- h is the smoothing length

then the density estimate is in the form:

$$\rho_i = \sum_{j=1}^{N} m_j W(|r_{ij}|, h_i) \tag{3.11}$$

For more details see [Spr05a], [SYW00] and [Inc11i].

3.4 Periodic Boundary Conditions

Periodic Bundary Conditions are used to simulate large systems by only modelling a small part of it that is far from the side of the system. It represents a perfect tiling of the simulated box, and that means that if a particle exits the box on a side, it will reaper on the opposed side with the same velocity. Take for example Figure 3.7 in which a particle moves from red to green to blue through the boundary.

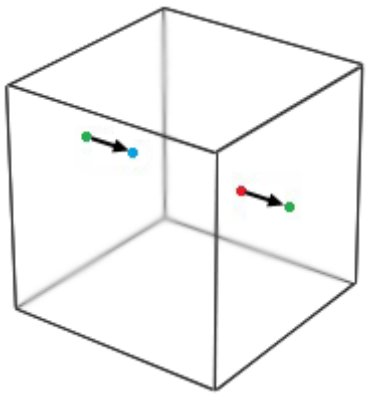

Figure 3.7: Example of a particle moving in a periodic box from red to blue.

The same rule applies to the gravitational potential, if red and blue were 2 different particles, then the red particle would feel a force pointing towards the interior of the box and a force poining towards the side of the box, generated by the blue particle. The forces through the boundaries are computed using a method called *Ewald summation* and for more details about the implementation in GADGET-2 see [Spr05a].

Simulations in periodic boxes allow us to study cosmological structure formation in a fairly small region compared to the size of the universe. This means that if we keep the box size constant and increse the number of simulated particles, we increase the resolution of the simulation thus, having a more exact picture of the density distribution. Simulations in periodic boxes rely on the *Cosmological Principle* that states: **On the largest cosmic scales, the Universe is both homogenous and isotropic. Homogeneity** means that there is no preferred location in the Universe. That is, no matter where you are in the Universe, if you look at the Universe, it will look the same. **Isotropy** means that there is no preferred direction in the Universe. That is, from your current location, no matter which direction you look, the Universe will look the same. Figure 3.8 is an all-sky map of the locations of objects detected by radio telescopes that reveals a uniform appearance of the Universe supporting the Cosmological Principle.

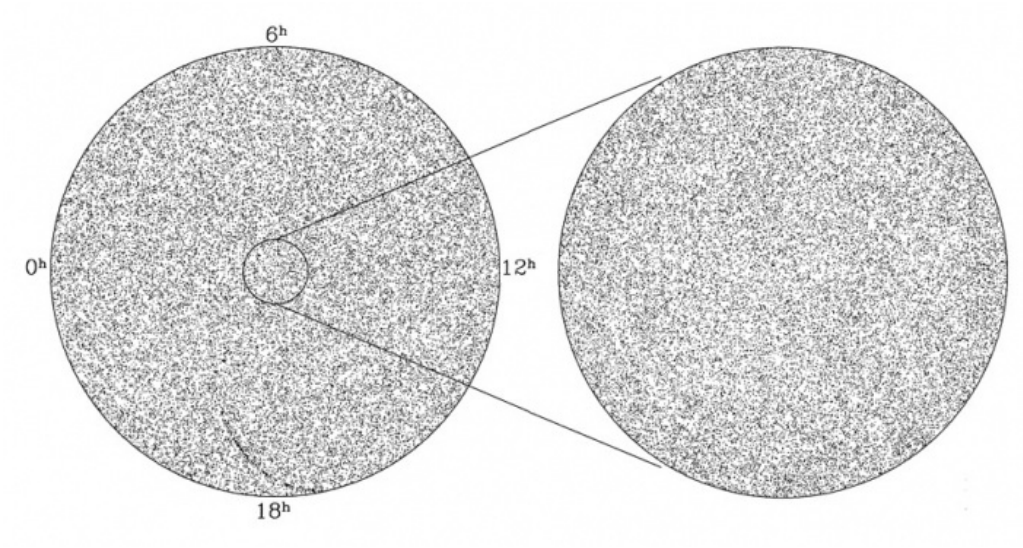

Figure 3.8: All-sky map of objects detected by radio telescopes in the Universe.

3.5 Comoving Coordinates

3.5.1 Measuring Distance in the Universe

Firstly, let's discuss about measuring distances in space. The universe is so incredibly large, that measuring it's size using units like the metre is not very usefull. The difference clearly is very big between 10^{10} metres and 10^{20} metres but in fact how big is that? One usefull measurement unit is the *astronomical unit* or AU equal to the mean distance from Sun to Earth.

- 1 AU = 149.597.870.700($\sim 149,6 \times 10^9$) metres or $\sim$ 8 light minutes.
- 1 ly(light-year) = 9.460.730.472.580.800($\sim 9.46 \times 10^{15}$) metres.

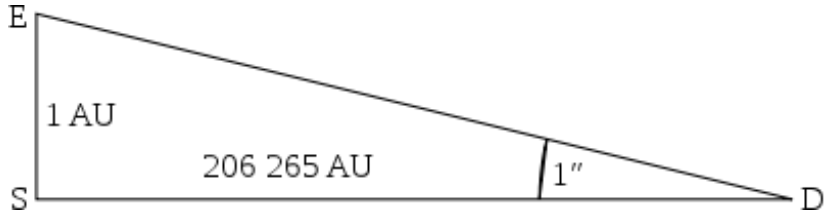

Figure 3.9: Diagram used to calculate the value of a parsec.

The AU is still not large enough because we know that the nearest known star to the Sun is Proxima Centauri at 4.2 light-years (3.97×10^{16}) metres. We also can't

use the light-year, because it's not particulary relevant when observing a distant star from Earth. So, the astronomic community uses a unit of length called a *parsec(pc)*. It's value can be computed using the diagram in Figure 3.9. ES is the distance between the Earth and the Sun and the angle SDE is one arcsecond ($\frac{1}{3600}$) of a degree. Then:

- 1 pc = 3,085678×10^{16} metres = 3,262564 ly

Using the parsec Proxima Centauri is 1,29 pc away, the center of the Milky Way is about 8.000 pc away, the Milky Way is about 30.000 pc across, and the Andromeda Galaxy(M31) is slightly less than 800.000 pc(0.8 Mpc) away from Earth. In a later chapter we will discuss a simulation in a Periodic Box of 50 Mpc, that means if we plot the particles on a 600 × 600 pixels image, then the distance between the Milky Way and the Andromeda Galaxy would be around 10 pixels. The diameter of the observable universe is at least 14 Gpc. [Inc11g]

3.5.2 The Expanding Universe

In the 1920s the American astronomer Edwin Hubble using the newly built Hooker Telescope in California discovered that some spiral shaped nebulae are too distant to be part of the Milky Way and are, in fact, entire galaxies outside our own. Furthermore, he discovered that the galaxies were moving away from us and also away from each other and the farther they were, the faster they moved. This discovery was later named the *Hubble's law* [Inc11d]. The velocity is determined using a method called spectroscopy which measures the Doppler shift of type *1a* supernovae(called standard candles) located in distant galaxies. Because they are moving away, the light appears to be redshifted(shifted towards the red side of the color spectrum). However, the cosmological redshift is caused by the expansion of space which the light travels and not by the Doppler effect (because the speed of light is constant).

Therefore, we can use the redshift as a mean for determining the distance to another galaxy and also it's speed. If we take into account the speed of light then we can also say that the bigger the redshift of an object the more in the past we are watching. The highest redshift of an object observed is z=8.2 corresponding to an age of the Universe of 640 million years. The cosmic microwave background radiation [Inc11b] has a redshift of z=1090.89 corresponding to age of the universe of only 436 million years, which is the farthest event in space and time that we can detect. The age of the universe is estimated to 13.75 ± 0.11 billion years. Below, in Figure 3.10 a plot of nearby observed galaxies on the redshift distance scale with Earth in the center, showing also the large scale structure of the Universe.

The distance can be measured using the *comoving distance* or the *proper distance*. The proper distance to a distant object is the actual distance to that specific object at a specific moment in time. It usualy grows due to the expansion

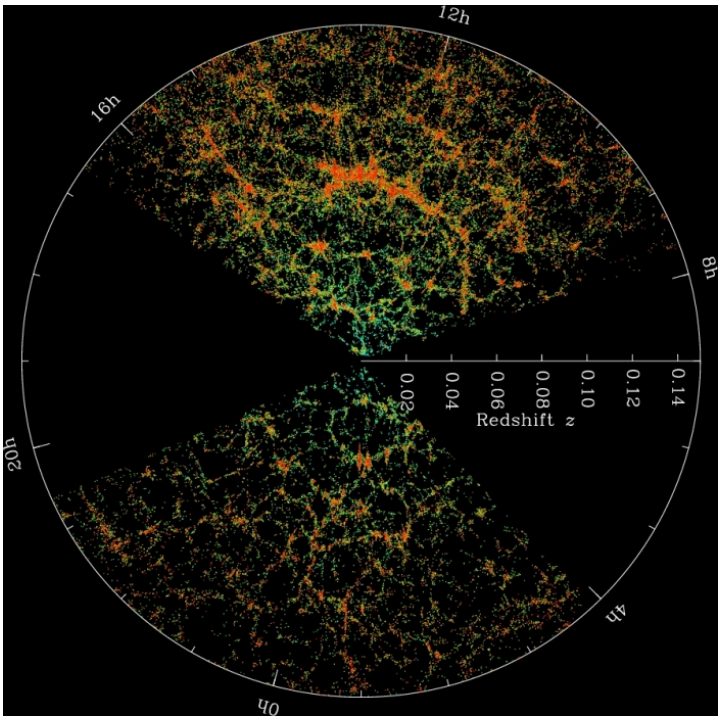

Figure 3.10: Plot of nearby observed galaxies on the redshift distance scale, with Earth in the center.

of the universe. The comoving distance factors out the expansion of the universe, giving a distance that doesn't change over time, though it is defined to be equal to the proper distance at the present time.

In the ΛCDM model the expansion(caused by *dark energy* or *vacuum energy*) is accelerating. The expansion rate is given by the percentage of dark energy and matter in the universe. One can use Edward L. Wright's cosmological calculator, publicly available on the Internet, to calculate the age of objects at different redshifts using different values for the hubble constant and the density for matter and vacuum [Wri06]. GADGET-2 can run simulations using both comoving distances and proper distances.

For further information see the following references: [Wri09], [Inc11h], [Inc11f], [Inc11d], [LD05].

In the next chapter we will proceed to matters concerning simulations with GADGET-2.

Chapter 4

Hardware and Software Configuration

In this chapter we will discuss the hardware and software used for running N-Body simulations with GADGET-2. We will start with the hardware configuration of the NCIT Cluster and after, the steps needed towards our first simulation with GADGET-2.

4.1 The NCIT Cluster

The NCIT Cluster is located at "Politehnica" University of Bucharest and it's a Beowulf cluster made out of Intel Xeon and AMD Opteron processors consisting of 456 cores in total. The cluster is currently used in research and teaching purposes by teachers, PhD students, grad students and students. For access and

Model	Processor Type	Sockets/ Cores	Slots	Memory	Queue
IBM HS21, 28 nodes	Intel Xeon E5405, 2 GHz	2/8	224	16 GByte	ibm-quad.q
IBM HS22, 4 nodes	Intel Xeon E5630, 2.53 GHz	2/16	64	32 GByte	ibm-nehalem.q
IBM LS22, 14 nodes	Amd Opteron 2435, 2.6 GHz	2/12	168	16 GByte	ibm-opteron.q

Table 4.1: NCIT Cluster x86 queues.

login ID, as well as other information, please refer to [Pro11]. We will access the cluster using it's Front End Processor machine (`fep.grid.pub.ro`) via ssh. The

FEP and the nodes run RHEL supporting both x86 and x86_64 architectures on the 2.6.18 Linux kernel and offer support for compiling and submitting jobs to the cluster. The jobs are submitted using the Sun Grid Engine on three main queues. In Table 4.1 is a short description of the available queues. For more information and advanced usage, please see [Sun07], [Sun08].

For our research we will use only the opteron nodes because they have the most powerfull interconnect between themselves and the LustreFS distributed file system nodes: 40Gbps InifiniBand. Because GADGET-2 is a massively parallel code, the performance strongly depends on the performance of the links between the nodes. In Appendix B, Table 9 we detail the hardware configuration of the Opteron nodes [IBM09].

4.2 Requierments for GADGET-2

In this section we will list the software packages needed to install and run GADGET-2. We stress the fact that for a proper functionality a modern Linux system is strongly advised.

Because the NCIT Cluster is a system shared by many users, one should compile all the packages in it's own home(using the **--prefix** flag of the configure script) and then modify the `PATH, LIBRARY_PATH and LD_LIBRARY_PATH` environment variables to use the new packages. A complete list of the installed packages at Appendix B, Table 12.

4.2.1 Compiler - GCC

A compiler is needed to build the dependencies and other packages. Your system must have an installed compiler such as GCC or SunStudio. The FEP has installed by default the 4.1.2 version of GCC that came with the Linux distribution, a somewhat old version. We used it to compile the 4.6.0 version of GCC which was released on 26^{th} of April, 2011. Then with the new GCC we compiled all the dependencies listed below. The difference in performance was about 5% between the old and the new compiler especially on the versions with compiler optimizations. You should build the newest version of the preferred compiler, and from the experience gathered so far with GADGET-2, GCC is the best suited. We will not cover the steps involved in installing GCC, but a thorough tutorial can be found at `http://gcc.gnu.org/install/`.

4.2.2 Message Passing Interface - OpenMPI

GADGET-2 is a massively parallel code written with MPI, therefore an MPI implementation is needed. We recommend the newest OpenMPI implementation which we tested in two flavours: with and without OpenIB(InfiniBand) support.

Because the opteron nodes have an InfiniBand interconnect, the overhead involved with the TCP/IP protocol can slow down the communication. We will make a performance comparison of the two flavours of the 1.5.3 beta version of OpenMPI.

In order to compile OpenMPI with OpenIB support the following flags need to be passed to the configure script: **--with-sge --enable-openib-control-hdr-padding --enable-openib-connectx-xrc --enable-openib-rdmacm --enable-btl-openib-failover --with-openib**. Please consult Appendix B, Table 12 for the dependencies needed. Also in some cases in the **bin/** directory where GCC is installed, symbolic links should be made named **f77, f95, g77** pointing to **gfortran**.

4.2.3 FFTW - Fastest Fourier Transform in the West

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data. We use the 2.1.5 version of the package because it's the latest with MPI parallel transforms support. Pleasce check `http://www.fftw.org/`.

One should compile FFTW with both single-precision and double-precision support. This is done by building and installing it twice, first with the **--enable-mpi --enable-type-prefix** configure flags and second with **--enable-mpi --enable-type-prefix --enable-float**. Don't forget to run **make clean** between the 2 builds.

4.2.4 GSL - GNU Scientific Library

The GNU Scientific Library is a numerical library for C and C++ programmers available at `http://www.gnu.org/software/gsl/`. The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total with an extensive test suite.

No special flags are requiered for building gsl.

4.2.5 HDF5 - Hierarchical Data Format

Version 5.0 is requiered: `http://www.hdfgroup.org/downloads/index.html`. This optional library is only needed when one wants to read or write snapshots in HDF5 format. It is possible to compile and use GADGET-2 without this library.

4.2.6 SZIP Compression library

SZIP is requiered by HDF5, to compress files in the hdf5 file format, available at http://www.hdfgroup.org/doc_resource/SZIP/. Take note that most snapshot visualization tools don't support the hdf5 format.

4.3 Installing GADGET-2

GADGET-2 can be downloaded from the address http://www.mpa-garching.mpg.de/gadget/, currently at version 0.7. It supports tweaking with compile-time options depending on the type of simulation we wish to run, therefore this step needs a little detailing.

First, we need to edit the paths to the needed libraries (fftw, gsl, hdf5, szip), the compiler and the compile flags. We need to define a new system type(considering that all libraries were installed in the same location), see an example below in Listing 4.1.

Listing 4.1: Configuring SYSTYPE in GADGET-2's Makefile

```
1 SYSTYPE="NCIT"
2 ifeq ($(SYSTYPE),"NCIT")
3 CC       =  mpicc
4 OPTIMIZE =  -O3 -m64
5 GSL_INCL =  -I$HOME/opt/include
6 GSL_LIBS =  -L$HOME/opt/lib -lz
7 HDF5LIB  =  -lhdf5
```

Second, we need to decide what kind of simulation we want to run. In Listing 4.2 is a list of the most common options used with values configured for a simulation of 2 colliding galaxies.(The options with # in front are not set)

Listing 4.2: Configuring the simulation type in GADGET-2's Makefile

```
1 #OPT   +=  -DPERIODIC
2 OPT    +=  -DUNEQUALSOFTENINGS
3 OPT    +=  -DPEANOHILBERT
4 OPT    +=  -DWALLCLOCK
5 #OPT   +=  -DPMGRID=256
6 #OPT   +=  -DDOUBLEPRECISION_FFTW
7 OPT    +=  -DSYNCHRONIZATION
8 OPT    +=  -DNOSTOP_WHEN_BELOW_MINTIMESTEP
```

A full list of options can be found in Appendix B, Listing 1. For details about each option, please see GADGET-2 user's guide: [Spr05b]. Whenever the Makefile is changed, a recompilation of the code is necessary. This can be achieved by running `make clean`, followed by `make`. Further code options of GADGET-2 are controlled by a parameter file, which is described in the next section.

4.4 Running GADGET-2

4.4.1 Configuring a Simulation

Before running a simulation with GADGET-2 we need to cover some steps. First, we need to have an initial state of our system, which will evolve durring the simulation, called the *initial conditions file(IC)*. Generating initial conditions will be covered in a later section. Second, there are some parameters defined in a *parameter file* given as an argument to the program. Below in Listing 4.3 are the most important parameters in the parameter-file. For a complete list see Appendix B, Listing 2 and [Spr05b].

Listing 4.3: Important simulation parameters in the parameter file

```
1 InitCondFile          ./small.gadget
2 OutputDir             ./
3 OutputListFilename    ./output_list.txt
4 TimeLimitCPU          360000  % in seconds
5 OutputListOn          1
6 TimeBegin             0.0
7 TimeMax               1.0
8 TimeBetSnapshot       0.01 %time between snapshots
9 TimeOfFirstSnapshot   0.00
```

Before running the simulation it is important to make sure all the paths to the requiered libraries are in the `LD_LIBRARY_PATH` and `LIBRARY_PATH` environment variables (you can check this with the `ldd Gadget2` command). We advise to use the `$HOME/.bashrc` file for defining the variables, please see the example in Appendix B, Listing 3.

4.4.2 Running a Simulation on the NCIT Cluster

Finally, in order to start the simulation we need to run the following command:

```
mpirun -np <N_PROC> ./Gadget2 <PARAMETER_FILE> [1|2]
```

where:

- `<N_PROC>` is an integer meaning the number of MPI processes we'd like to run GADGET-2 on.
- `<PARAMETER_FILE>` name of a text file with a special syntax containing parameters for the run.
- `[1|2]` optional parameter for restarting the simulation using restart files(1) or a snapshot(2) see [Spr05b].

Of course this will make the simulation run on the FEP which is forbidden. In order to benefit from the computational power of the cluster we need to submit our job using the Sun Grid Engine.

Supose we wish to run a simulation which has the IC file and the param file in `sim-dir`. Firstly, we need to create a script file, say `s.gadget2.sh`:

Listing 4.4: Example of a running script

```
1 cd sim-dir/
2 mpirun -np $1 ../Gadget2 parameter-file.param
```

Then we need to run the `qsub` command, below an example for 36 slots:

```
qsub -q ibm-opteron.q -pe openmpi 36 -cwd ./s.gadget2.sh 36
```

After the submit we can view the status of the job using the `qstat` or `watch qstat` commands. In the *State* column, `qw` stands for *queue wait* and `r` for *running*. If we would like to watch the output to STDIN or STDERR we must print the files: `s.gadget2.sh.o<job_id>` and `s.gadget2.sh.e<job_id>` (`job_id` is displayed using `qstat`).

This command will reserve 36 slots for us on the opteron queue but it will not guarantee the fact that we use only 3 nodes, even when 3 free nodes are available. Because of this some of the mpi processes can get submitted to a busy node with only a few available slots, this translating in lower performance.

For manually assigning the working nodes, we must first find out which nodes are free, and what is their load. For this we can use a script containing the following lines:

Listing 4.5: free_nodes.sh: Script for discovering the free cluster nodes

```
1 # takes as first arg 'quad', 'opteron' or 'nehalem'
2 qstat -F |head -n 1
3 qstat -F |grep $1-wn |grep $1.q
```

An output of this script can be found in Appendix B, Listing 4. After seeing the output we decide that we want to run on the opteron-wn03, opteron-wn04 and opteron-wn05 nodes, so we use the following command:

```
qsub -q ibm-opteron.q@opteron-wn03*\|opteron-wn04*\|opteron-wn05*
-pe openmpi 36 -cwd ./s.gadget2.sh 36
```

If we want to see the resources used on the nodes, and which nodes are running our jobs, we can use the following commands:

```
qstat -f |grep $USER -B 2
qstat -F |grep opteron -A 14 |grep mem_used |head -n 14
```

Chapter 5

Simulations With GADGET-2

In this chapter we will discuss more specifically the types of N-Body simulations available through GADGET-2. Regardless of the type all simulations require an initial state of the system called *Initial Conditions*, and according to the parameter file the results outputted are intermediate snapshots of the system between the start and the end of the simulation. The snapshots allow us to visualize the system's evolution in time. The output frequency of the snapshots can be controlled via the `TimeBetSnapshot 0.1` and `TimeOfFirstSnapshot 0.3` options in the parameter file or by a special file configured in the parameter file as follows: `OutputListOn 1` and `OutputListFilename ./output.txt`. The file contains the output times, one on each line. The output time is the internal simulation time, and for cosmological simulations Time = 1.0 for Redshift z = 0.0. By default GADGET-2 writes a snapshot at the end of the simulation.

Also as a result of the simulation several code diagnostic files are created (see [Spr05b] for details):

- `InfoFile:` A list of all the timesteps of the simulation
- `TimingsFile:` Statistics about the performance of the gravitational force computation by the tree algorithm
- `CpuFile:` Statistics about the total CPU consumption measured in various parts of the code while it is running. See Appendix B, Table 13
- `EnergyFile:` Statistics about the total energy of the system

5.1 Types of Simulations

In the GADGET-2 User Guide [Spr05b] 10 types of simulations are detailed, all of them made out of variations of 5 computational methods:

- Tree – Force computation using only the tree method
- TreePM – Long-range force is computed with the PM
- SPH – The simulation includes SPH particles
- Boundaries – Periodic or vacuum boundaries
- Comoving Integration – Simulation runs in comoving coordinates rather than proper coordinates(newtonian space). The time of the simulation is measured as a redshift value.

We will study 2 classes of simulations: classical N-Body simulations in newtonian space and Cosmological simulations in newtonian and comoving space. For Cosmological simulations we need special Initial Conditions that mimic an expanding universe. The type of simulation is chosen by options in the parameter file(PF) and in the Makefile(MF), see Table 5.1:

Simulation Type	PERIODIC (MF)	PMGRID (MF)	Comoving Integration On(PF)
Newtonian space, Tree, (SPH)	not set	not set	0
Newtonian space, TreePM, (SPH)	not set	set	0
Cosmological, Newtonian space, Tree, (SPH)	not set	not set	0
Cosmological, Comoving coordinates, Tree, (SPH)	not set	not set	1
Cosmological, Comoving periodic box, Tree, (SPH)	set	not set	1
Cosmological, Comoving coordinates, TreePM, (SPH)	not set	set	1
Cosmological, Comoving periodic box, TreePM, (SPH)	set	set	1

Table 5.1: Selecting the simulation type.

Cosmological simulations should be configured with comoving coordinates. Otherwise, the particles will expand over time making visualization difficult because of the increasing range of the positions of particles. This is also the reason why cosmological simulations in a periodic box are available only with comoving coordinates.

5.2 Building Initial Conditions

Building Initial Conditions for N-Body simulations that converge into significant results is not a trivial task. For cosmological models one should take in account the microwave background anisotropy. COSMICS [MB95] is a package of Fortran programs useful for computing transfer functions and for gaussian random initial conditions for nonlinear structure formation simulations of cosmological models. For one of our case study we used another initial conditions generator called N-genIC, made available by Volker Springel via the DEISA Benchmark Suite: `http://www.deisa.eu/science/benchmarking`.

Most of the initial conditions generator use the Zel'dovich approximation as a way of following the initial evolution of cosmic structure when density perturbations become too large for the Cosmological perturbation theory to be valid. It was invented in the 1970s by Yakov Zel'dovich, who used it to show that the first structures to form after the Big-Bang were large sheet-like structures which became known as Zel'dovich pancakes. The cosmological perturbation theory applies in situation in which the universe is predominantly homogeneous, today only on the largest scales [Inc11c].

In this section we will detail the file format and a simple program for writing initial conditions with particles at random positions and velocities.

5.2.1 The GADGET Standard File Format

This is the standard file format, compatible with GADGET-1 and GADGET-2. In order to use this format it must be configured in the parameter file of the simulation via the options: `ICFormat 1` and `SnapFormat 1`. The file is made out

Block ID	Block content
HEAD	Header
POS	Positions
VEL	Velocities
ID	Particle ID's
MASS	Masses(only for particle types with variable masses)
U (sph*)	Internal energy per unit mass
RHO (sph*)	Density
POT (mf*)	Gravitational potential of particles
ACCE (mf*)	Acceleration of particles
ENDT (mf*, sph*)	Rate of change of entropic function of SPH particles
TSTP (mf*)	Timestep of particles(only when enabled in makefile)

Table 5.2: Blocks in the GADGET standard file format.

of blocks, and each block is delimited at the beginning and at the end by a 32 bit unsigned integer having the value of the blocks length in bytes. In Table 5.2

is a list of all the available blocks (The ones with sph*, are only available for SPH particles. The ones with mf* are only available when enabled in Makefile). Besides the ID block which containes 32 bit unsigned int values and the HEAD block which contains a struct SnapshotHeader, all block contain 32 bit float or 64 bit double (if the DOUBLEPRECISION is activated in the Makefile). For details see Appendix B, Table 14.

GADGET-2 has 6 types of particles, each interact with each other and with themselves. The *Gas* particles are special SPH particles, the others are ordinary N-Body particles. Each particle support different softening values (UNEQUALSOFTENINGS must be activated in the Makefile). A table with their name and id below, Table 5.3:

Type	Symbolic Type Name
0	Gas
1	Halo
2	Disk
3	Bulge
4	Stars
5	Bndry

Table 5.3: Different particles types in GADGET-2.

In Appendix B, Listing 5 is an example of a program written in C for generating trivial cosmological initial conditions in the GADGET default file format, featuring only one type of particles. Using this example for writing snapshot files, one can develop tools for reading the header information of a snapshot file and the particle data.

5.2.2 Visualization of Snapshot Data

In order to plot the particles positions we used IFrIT [Gne], a general purpose visualization software for Windows with an interactive GUI. Using the GADGET extension we can read simulation snapshot files separating particles by their type. After setting the angle of the *scene* and the plotting details for each particle type, such as size, color and transparency, IFrIT can save the image as a **png** file. Using the *animation* capability, it can load the snapshot files from the same folder incrementally, and save a picture of the scene, but only if they are named according to the rule **<name>_xxxx.bin** where **xxxx** is a number from **0000** to **9999**. One can use the script in Appendix B, Listing 6 to rename the snapshots.

After having each frame stored in a png file we used AviSynth(a script like programming language and tool for movie processing) to compile the images into a video clip. In order to encode the clip in a portable format such as MPEG-4 we used AviDemux together with AVSProxy_gui.

Chapter 6

Case Studies

In this chapter we will analyze GADGET-2's performance for a series of test simulations ranging from 40^3 to 768^3 particles. We will plot the Speedup (6.1) and Efficiency (6.2) for each simulation with respect to the number of processors.

$$S_p = \frac{T_1}{T_p} \tag{6.1}$$

$$E_p = \frac{S_p}{p} \tag{6.2}$$

where:

- p is the number of processors
- T_1 is the execution time of the sequential algorithm
- T_p is the execution time of the parallel algorithm with p processors

For each simulation also the longest and shortest running times are mentioned, along with the memory consumption. We stress the fact that this tests were carried on the opteron queue, which has 12 processors per cluster node, for OpenMPI with and without OpenIB support. The steps in the number of processors respect the number of processors per node, first measuring the scalability for one node and then adding one node at a time: 1, 2, 4, 6, 8, 10, 12, 24, 36, 48, 60, 72, 84, 96, 108, 120, 132. For the formulas needed for calculating the memory consumption of the code see Appendix B, section 7.2.

6.1 A Pair of Colliding Galaxies

For the first case study we choose to test a purely collisionless simulation which runs two disk galaxies into each other, leading to a merger between the galaxies. Each galaxy is composed out of a stellar disk (10.000 *disk* particles) and a

massive and extended dark matter halo (20.000 *halo* particles). The simulation is runned in Newtonian space with the Tree algorithm and vacuum boundaries. In Figure 6.1 is plotted the initial and final state of the system. For a video of the simulation see [Spi11].

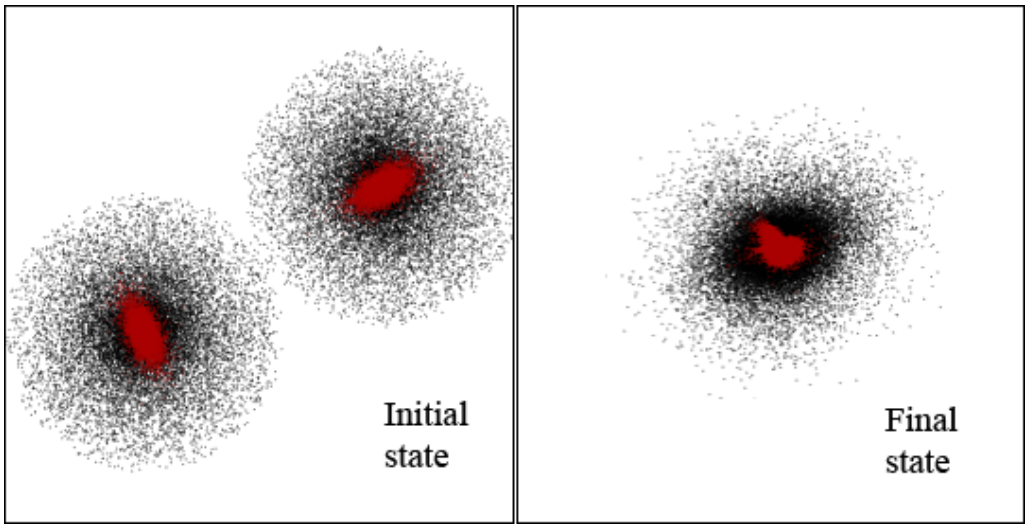

Figure 6.1: The Initial state (simulation-time 0,0) and Final state (simulation-time 1,0) of the "Galaxies" simulation. *Disk* particles in red and *halo* particles in black.

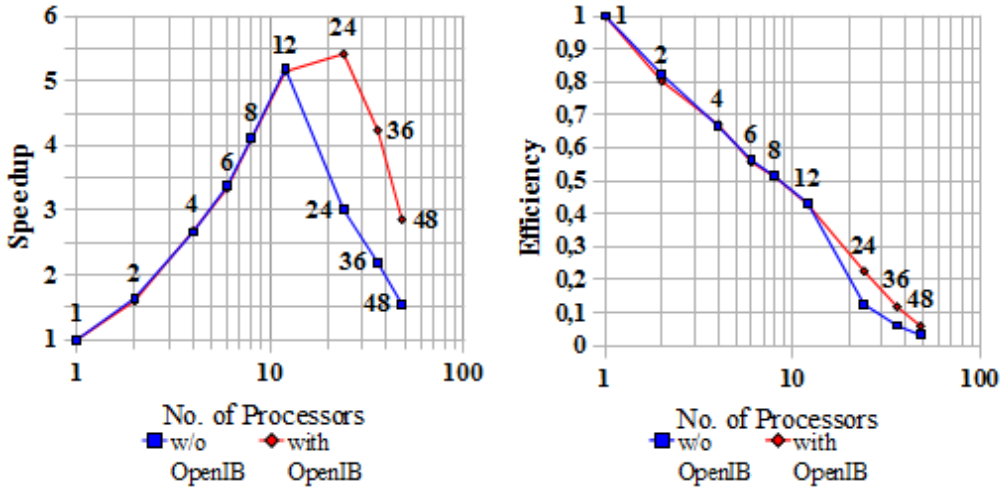

Figure 6.2: Speedup and Efficiency for the "Galaxies" simulation.

In Figure 6.2, we plot the Speedup and Efficiency of GADGET-2 for this small problem. A snapshot file has a size of only 1,7 MB. The total time for the slowest

run (1 processor) is **520,5** seconds and **1,01** seconds/step. For the fastest run (24 processors) the total time is **95,43** seconds and **0,19** seconds/step. Between the two we obtain a maximum Speedup of **5,42**. For all the values, see Appendix A, Table 1.

The total memory consumption of the simulation is about 10 MB for the particles plus 25 MB communication buffer for each processor.

6.2 Cosmological Formation of a Cluster of Galaxies

The second case study is a problem that uses collisionless dynamics in an expanding universe. It is a small cluster simulation that uses vacuum boundaries, comoving integration and the TreePM algorithm on a 128^3 mesh. The total number of particles is 276.498. In a central high-resolution zone there are 140.005 *halo* particles, sorrounded by a boundary region with two layers of different softening, the inner one containing 39.616 *disk* particles, and the outer one 96.877 *bulge* particles. In Figure 6.3 is plotted the initial and final state of the system. For a video of the simulation see [Spi11].

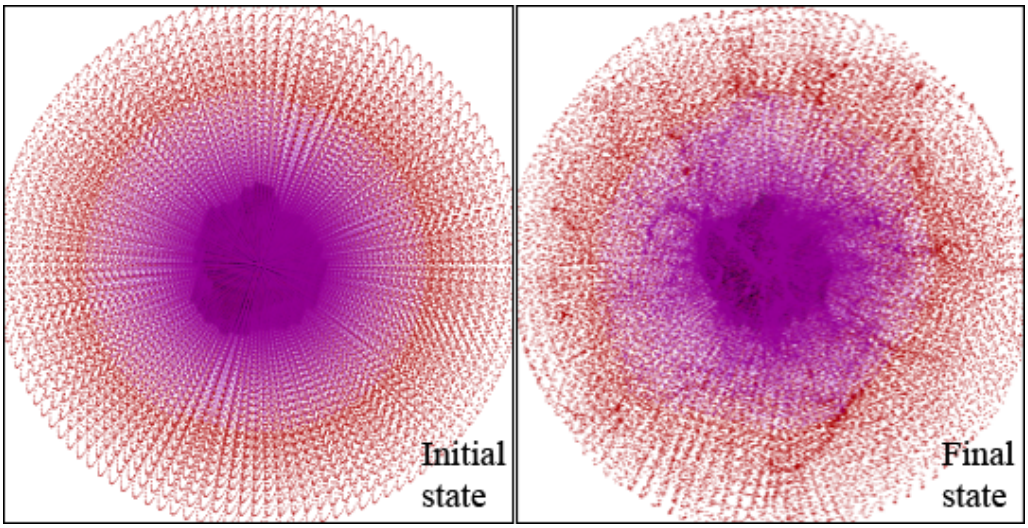

Figure 6.3: The Initial state (redshift 23) and Final state (redshift 2,33) of the "Cluster" simulation. *Halo* particles in black, *disk* particles in red and *bulge* particles in purple.

In Figure 6.4, we plot the Speedup and Efficiency of GADGET-2 for the "cluster" problem. A snapshot file has a size of about 8 MB. The total time for the slowest run (1 processor) is **2369,11** seconds and **5,68** seconds/step. For the fastest run (36 processors) the total time is **189,9** seconds and **0,46** seconds/step.

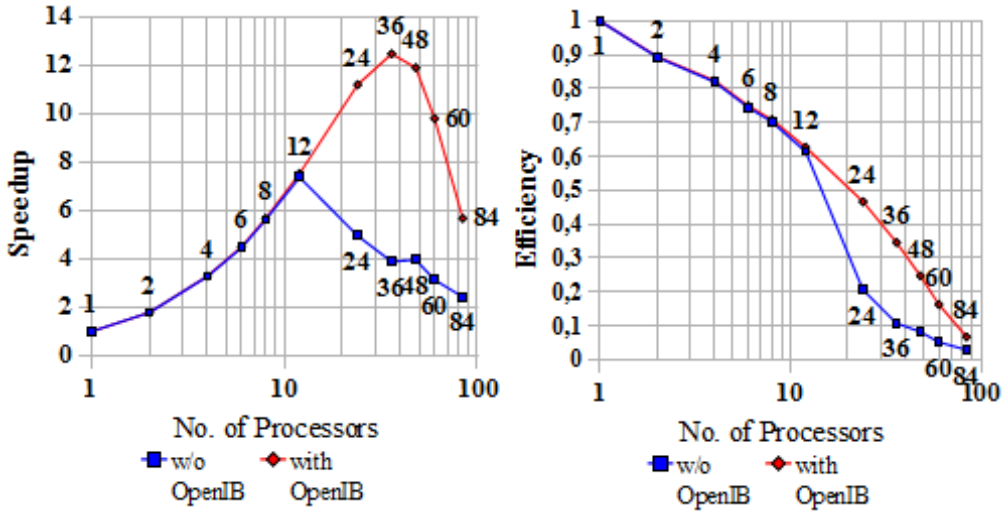

Figure 6.4: Speedup and Efficiency for the "Cluster" simulation.

Between the two we obtain a maximum Speedup of **12,48**. For all the values, see Appendix A, Table 2.

The total memory consumption of the simulation is about 53 MB for the particle storage and 268 MB for the TreePM in total plus 30 MB communication buffer for each processor.

We already observe a trend that indicates the bigger the number of particles, the higher the speedup and the efficiency for a greater number of processors. Also the critical need for a performant interconnect, when using multiple nodes, as an argument see the difference between MPI on the 40Gbps InfiniBand with and without TCP/IP encapsulation.

6.3 Large-scale ΛCDM Structure Formation Including Gas

The third case study is a problem that follows structure formation in a ΛCDM universe. Only adiabatic gas physics is included, and the minimum temperature of the gas is set to 1000 K. This example uses a periodic box of size $(50h^{-1}\text{Mpc})^3$, comoving integration, the TreePM algorithm on a 128^3 mesh, and SPH particles to simulate the gas. The total number of particles is 65.536 split equal into *gas* and *halo* particles. *Gas* particles are put at the centres of the grid outlined by the dark matter particles. In Figure 6.5 is plotted the initial and final state of the system. For a video of the simulation see [Spi11].

In Figure 6.6, we plot the Speedup and Efficiency of GADGET-2 for the "Gas" problem. A snapshot file has a size of about 2 MB. The total time for the slowest run (1 processor) is **429,39** seconds and **0,70** seconds/step. For the

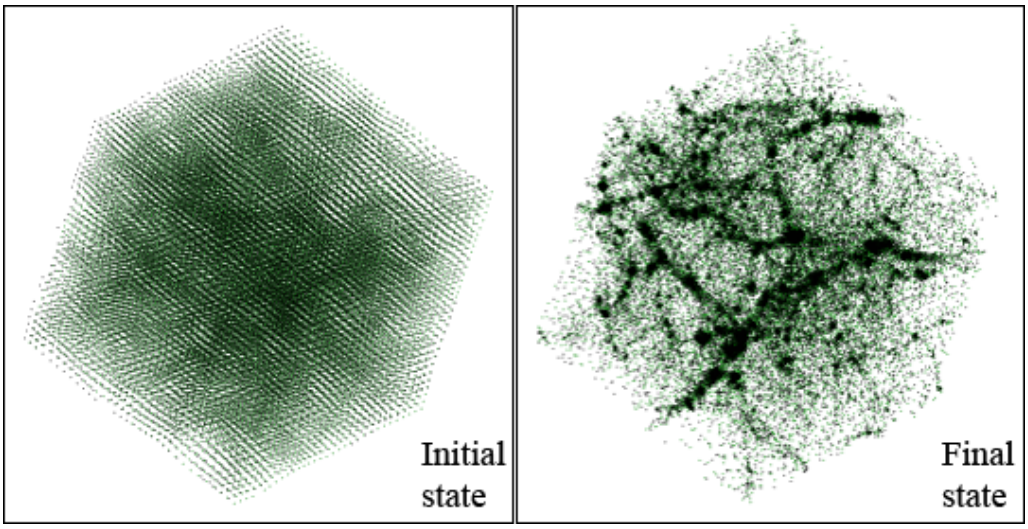

Figure 6.5: The Initial state (redshift 10) and Final state (redshift 0) of the "Gas" simulation. *Halo* particles in black, and *gas* particles in green.

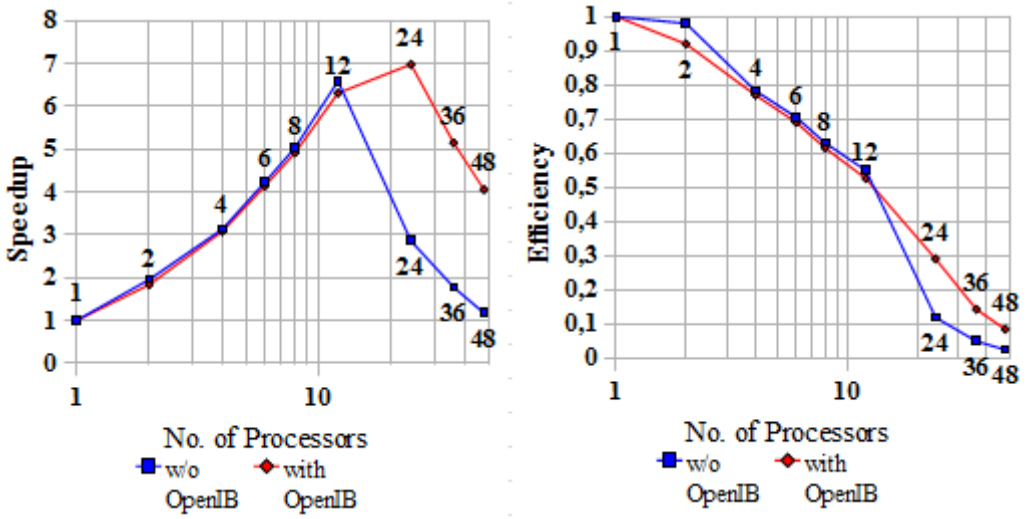

Figure 6.6: Speedup and Efficiency for the "Gas" simulation.

fastest run (24 processors) the total time is **60,02** seconds and **0,10** seconds/step. Between the two we obtain a maximum Speedup of **6,98**. For all the values, see Appendix A, Table 3.

The total memory consumption of the simulation is about 50 MB for the particle storage and 25 MB for the TreePM in total plus 80 MB communication buffer for each processor.

We again notice that the speedup drops when we run on more than 2 cluster

nodes, because of the communication overhead and synchronization between the steps. However this and the last two case studies were problems so small that running on a single cluster node on 12 cores would finish the simulation in a reasonable time, therefore not needing multiple cluster nodes. Furthermore, we observe the same trend in speedup for three different types of simulations.

6.4 Cosmological Density Fields

The last case study considers the formation of the large-scale strucutre of the universe generated by the evolution of cosmological density fields. The Initial Conditions are generated with N-GenIC, part of the DEISA benchmark suite. The simulation uses the TreePM algorithm on a mesh, double in size than the size of one dimension of the simulation, in a comoving periodic box and has only *halo* particles. The sizes of the simulations in Table 6.1.

Size	Mesh Size	Periodic Box Size	Snapshot Size	Duration (N_Proc)
128^3	256^3	$(25h^{-1}\text{Mpc})^3$	57 MB	2,29h (72)
192^3	384^3	$(37,5h^{-1}\text{Mpc})^3$	190 MB	5,26h* (132)
256^3	512^3	$(50h^{-1}\text{Mpc})^3$	449 MB	10,9h (132)
384^3	768^3	$(75h^{-1}\text{Mpc})^3$	1,5 GB	39,4h* (132)
512^3	1024^3	$(100h^{-1}\text{Mpc})^3$	3,6 GB	92,9h* (132)
768^3	1536^3	$(150h^{-1}\text{Mpc})^3$	12 GB	276,2h* (132)

Table 6.1: Size of the "Density Fields" simulation. Durations marked with * are estimates.

In Figure 6.7 we plot the initial and final state of the simulation for a total of 4096 steps. Because of the size of the simulations we runned the different versions of the simulation only for the first 4 steps, and then we calculated the time/step values using the last 3 steps, in order to avoid taking in consideration the time needed to read the Initial Conditions file (the size of a snapshot). A video of the 256^3 simulation can be found at [Spi11].

For this simulation we choose to run the simulation only with OpenMPI with OpenIB support, because of the overhead introduced by the TCP/IP stack, see Appendix A, Profiling. In Figure 6.8, the Speedup and Efficiency of GADGET-2 for the "Density Fields" problem. We can observe that increasing the size of the problem we greatly increase the parallel efficiency of the code. See the table with all the values at Appendix A, Table 4. Studying the results we observe that increasing the problem size by 8 we increase the Step duration by ~ 8 (128^3 vs. 256^3 and 192^3 vs. 384^3). Because of the size of the simulation, the 512^3 can run on a minimum of 2 cluster nodes, and the 768^3 on a minimum of 7 cluster nodes, so in order to calculate the Speedup we estimated(using the 8 factor) the step durations for the number of processors which we couldn't run the simulation for.

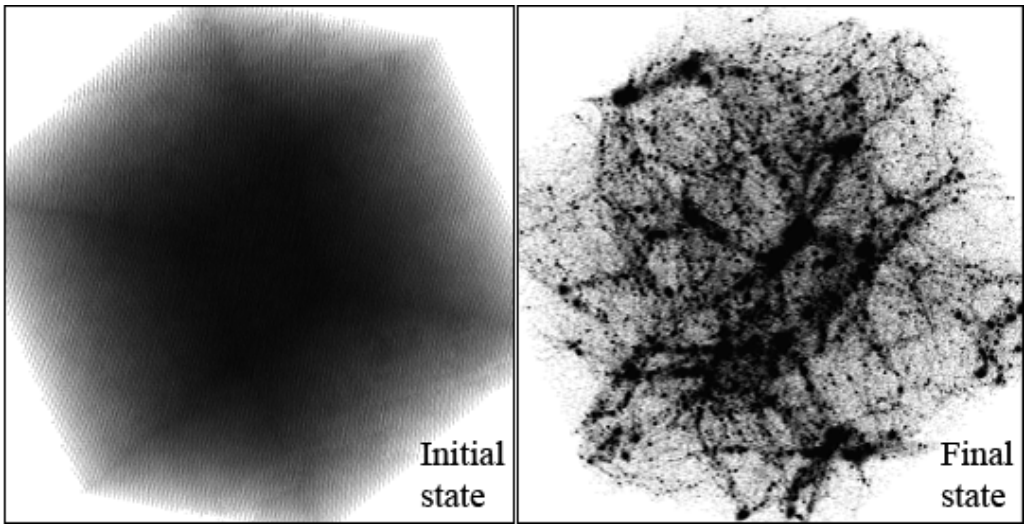

Figure 6.7: The Initial state (redshift 49) and Final state (redshift 0) of the 256^3 "Density Fields" simulation. 1.118.482 out of 16.777.216 particles in total.

We completed the simulations for the 128^3 on 72 processors and 256^3 on 132 processors choosing the number of processors by taking into account the highest efficiency observed for that particular problem size. Because of the way that N-genIC generates the problems, by tiling a 16^3 box and then computing the power spectum of the new system of particles, all simulations take 4096 steps to complete (from redshift 49 to redshift 0). We observed that the mean step

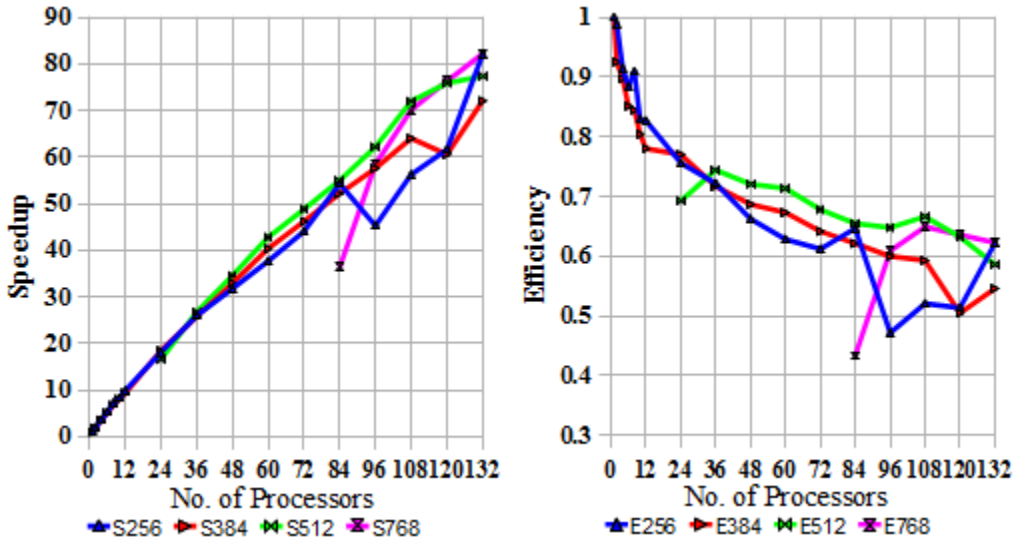

Figure 6.8: Speedup and Efficiency for the "Density Fields" simulations.

duration for the whole simulation is higher by a factor of $\sim 1,6 - 1,7$ than the mean step duration for the last 3 of the first 4 steps, because of the work-load imbalances that occur when the system evolves into more clustered regions. Based on this factor we approximated the values in the last column of Table 6.1 for the duration of the simulations on 132 processors.

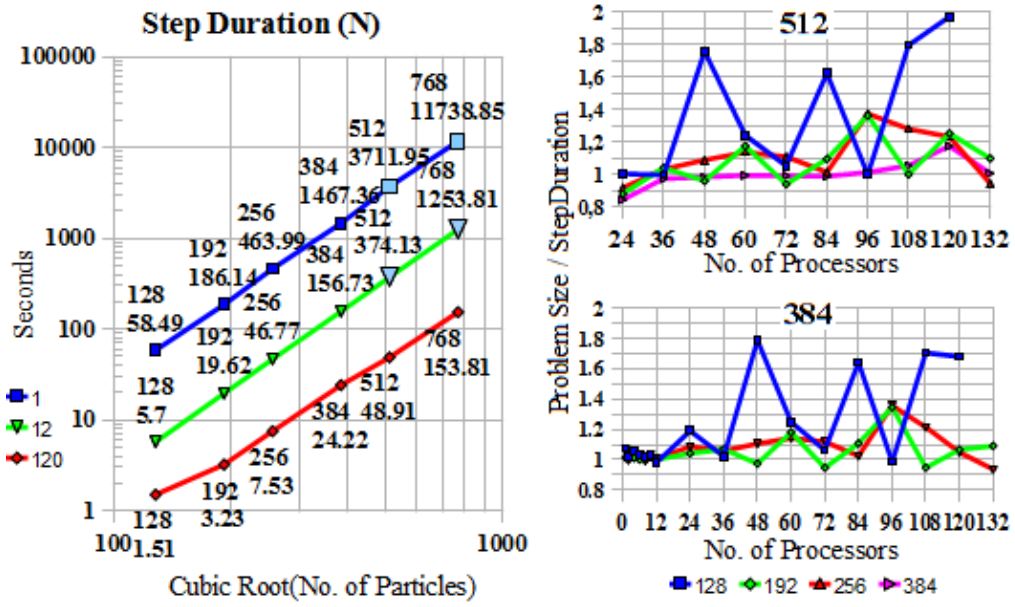

Figure 6.9: Step Duration analysis for the "Density Fields" simulations.

In Figure 6.9 in the graph on the left we plotted the step duration on logarithmic scales. Note that the values for 1 and 12 processors for the 512^3 and 768^3 are estimates. In the graphs on the right we plotted the results for the (6.3) and (6.4) formulas only for values obtained from experiments.

$$Val_{512}(S) = \frac{512^3}{S^3} / \frac{StepDuration(512^3)}{StepDuration(S^3)}, \quad S \in \{128, 192, 256, 384\} \tag{6.3}$$

$$Val_{384}(S) = \frac{384^3}{S^3} / \frac{StepDuration(384^3)}{StepDuration(S^3)}, \quad S \in \{128, 192, 256\} \tag{6.4}$$

A value above **1** means that the step duration is smaller than expected based on the proportion between the number of particles.

In Appendix A, Table 4 we also calculated the maximum memory consumption of the simulations. Note that a 100MB communication buffer per processor is not always necessary but it's useful to make it reasonably large when enough memory is available in order to avoid the crashing of the code caused by an insufficient buffer.

Chapter 7

Conclusions and Future Work

In this chapter we will present the thesis outlook, conclusions based on the case studies and recommendations for future work regarding N-Body Simulations with GADGET-2 on the NCIT Cluster.

7.1 Conclusions

The objective of this thesis was to set up the search for a "state of the art" tool that would employ modern techniques in solving N-Body problems, in a scalable and efficient manner. In the same time our aim was also to make an introduction to cosmological simulations and the algorithms employed, giving references for further studies. We chose the GADGET-2 simulation code, because it is developed according to the newest theoretical model of the universe, using very performant algorithms and also due to its renown in astrophysics simulation research. We succeeded in running complex simulations that prove not only the functionality of GADGET-2 but also the convergence and efficiency of the code for very large problems. Therefore, we are confident that this work can be used successfully as a starter's guide for N-Body Simulations with GADGET-2.

The simulations carried on in the Case Studies chapter, bring us to a series of very important conclusions regarding the performance of the code. Firstly, we noticed that we obtained a similar relation between the speedup and the number of processors / number of cluster nodes for different types of simulations, with and without Particle-Mesh and with and without SPH particles or periodic boundaries. Also, by making a comparison between the "Galaxy" simulation and the "Gas" simulation (which have an almost equal number of particles) we can state that the TreePM method is more efficient than the Tree method, however it uses more memory which can be a limiting issue for large simulations.

Secondly, we observed that the efficiency of the code si very closely related to the size of the problem. As a rule by increasing the size of the problem we also increase the efficiency of the code (for the same number of processors). For instance a simulation with 8×N particles runs only 6 or 7 times longer than the simulation with N particles, in the vast majority of cases.

When we speak about N-Body simulations we must first accept the fact that they take a long time to complete. Without accepting this fact we can end up running simulations on a too high number of processors that leads to very low efficiency of the code. Therefore, we state a trend in the range of the number of processors which we should run our simulations on. The **inferior limit** is given by the physical memory needed by the simulation(the minimum number of processors/cluster nodes), used for large simulations, and the **superior limit** is given by the efficiency(and the number of available processors), used for small simulations. We choose **0.4** the value of the efficiency under which we should not increase the number of processors even when we obtain a gain in speedup by doing so. Therefore, we recommend the following relation:

$$\textbf{superior limit} \leq \textbf{24} \times \textbf{inferior limit}$$

Running on the assumption that we have a cluster with nodes with 12 processors and 16 GB of RAM, then based on our experiments, the maximum time / Step (when we use 100% of the available memory – the inferior limit) is $\sim$ 240 seconds. **Close to the inferior limit when we double the number of processors we reduce the running time by $\sim$1.85**. See Appendix A, Table 5 and Table 6. In Table 7.1, stating the trend in necessary resources, we assume the cubic root of the mesh double than the cubic root of the particle number(giving a necessary of 260 bytes of memory/particle) and a relatively low 1,33 GB memory per processor.

Particle No./Mesh Size	Min. Nodes	Min. Proc.	Duration @ lower limit
128^3 / 256^3	1	1	$\sim$ 3 days
192^3 / 384^3	1	2	$\sim$ 7 days
256^3 / 512^3	1	4	$\sim$ 9 days
384^3 / 768^3	1	12	$\sim$ 12 days
512^3 / 1024^3	2	24	$\sim$ 17 days
768^3 / 1536^3	7	84	$\sim$ 18 days
1024^3 / 2048^3	17	204	$\sim$ 18 days
1536^3 / 3072^3	55	660	$\sim$ 18 days
2048^3 / 4096^3	130	1560	$\sim$ 18 days
3072^3 / 6144^3	439	5268	$\sim$ 18 days

Table 7.1: Resources necessary for large cosmologic simulations with GADGET-2. We assumed 1,6 work-load imbalance factor for the 192^3 to 3072^3 and 4096 steps / simulation.

By increasing the memory per processor and/or by reducing the mesh size, the duration at the lower limit can increase because we can run larger simulations on fewer processors.

Ultimately, the duration depends only on the average processing speed (part. /sec) value per processor. For the durations of the simulations larger than 768^3, we assumed a processing speed of 24.000 part./sec (Appendix A, Table 6).

7.2 Future Work

Taking into account the conclusions stated above, we consider further work on this thesis topic is advised and also necessary in order for it to bring valuable scientific results.

As a first step, we will run simulations on all the cluster nodes available at the NCIT site. This includes the opteron, quad and nehalem nodes. A new study in performance is necessary due to the latency and bandwidth issues regarding the interconnect between the different chassis.

Also the developing of an advanced Initial Conditions generator in colaboration with the Astronomical Institute of the Romanian Academy, could bring interesting results to the field. But running this new simulations requires a performant install of GADGET-2 capable of running problems with $N \geq 1024^3$, which we hope to achieve using the quad and opteron nodes together.

We also take in consideration the developing of a suite of post-processing tools capable of doing: slices through the snapshots of the system and density computations over the particle system. The latter is very useful for locating Stars and possible Black Holes.

Implementing a tool that can identify the self-bound halos in the system and their magnitude, such as Volker Springel's SUBFIND algorithm [SWTK01] should prove very useful in finding galaxies, clusters of galaxies, and giant voids in the simulated system.

Finally we will require a visualization tool with which an user can *navigate* through the particle system colored with respect to the density of the region and/or their smoothing lenghts, and *adjust* the "visibility" of the particles with respect to their magnitude. This might prove to be a very powerful aid in movies and presentations of the simulation's results.

Combining our work with the ideas exposed above, we will definitely obtain a very powerful platform for running Cosmological N-Body simulations. It will bring us closer to our final goal of making notable contributions to the scientific community.

Bibliography

[Bag02] J.S. Bagla. TreePM: a code for cosmological N-Body simulations. Paper, Harish-Chandra Research Institute, Chhatnag Road, Jhunsi, Allahabad 211019, INDIA, June 2002. [cited at p. 14]

[BH86] Josh Barnes and Piet Hut. A hierarchical O(NlogN) force-calculation algorithm. *Nature*, pages 446 – 449, December 1986. [cited at p. 10]

[Dub96] John Dubinski. A parallel tree code. Paper, Board of Studies in Astronomy and Astrophysics, University of California, Santa Cruz, CA 95064, USA, March 1996. [cited at p. 13]

[Fri09] Carsten Eie Friggard. G2X: GADGET2 Optimization Using the CUDA Architecture, http://sussi.megahost.dk/~frigaard/g2x/, November 2009. [cited at p. 5]

[Gne] Nick Gnedin. IFrIT User Guide, https://sites.google.com/site/ifrithome/. [cited at p. 29]

[IBM09] IBM. *IBM BladeCenter LS22 Product Guide*, February 2009. [cited at p. 21]

[Inc11a] Wikimedia Foundation Inc. Binary Star, http://en.wikipedia.org/wiki/Binary_star, May 2011. [cited at p. 8]

[Inc11b] Wikimedia Foundation Inc. Cosmic microwave background radiation, http://en.wikipedia.org/wiki/Cosmic_microwave_background, June 2011. [cited at p. 18]

[Inc11c] Wikimedia Foundation Inc. Cosmological perturbation theory, http://en.wikipedia.org/wiki/Cosmological_perturbation_theory, April 2011. [cited at p. 28]

[Inc11d] Wikimedia Foundation Inc. Hubble's law, http://en.wikipedia.org/wiki/Hubble%27s_law, June 2011. [cited at p. 18, 19]

[Inc11e] Wikimedia Foundation Inc. Lambda-CDM model, http://en.wikipedia.org/wiki/Lambda-CDM_model, May 2011. [cited at p. 2]

[Inc11f] Wikimedia Foundation Inc. Observable universe, http://en.wikipedia.org/wiki/Observable_universe, June 2011. [cited at p. 19]

[Inc11g] Wikimedia Foundation Inc. Parsec, http://en.wikipedia.org/wiki/Parsec, June 2011. [cited at p. 18]

[Inc11h] Wikimedia Foundation Inc. Redshift, http://en.wikipedia.org/wiki/Redshift, June 2011. [cited at p. 19]

[Inc11i] Wikimedia Foundation Inc. Smoothed-Particle Hydrodynamics, http://en.wikipedia.org/wiki/Smoothed-particle_hydrodynamics, April 2011. [cited at p. 15]

[Kne01] Knebe, A. and Green, A. and Binney, J. Multi-level adaptive particle mesh (MLAPM): a C code for cosmological simulations. *mnras*, 325:845–864, August 2001. [cited at p. 5]

[Kra99] Kravtsov, A. V. *High-resolution simulations of structure formation in the universe*. PhD thesis, NEW MEXICO STATE UNIVERSITY, 1999. [cited at p. 5]

[LD05] Charles H. Lineweaver and Tamara M. Davis. Misconceptions about the Big Bang, http://www.scientificamerican.com/article.cfm?id=misconceptions-about-the-2005-03, February 2005. [cited at p. 19]

[Lem] Lemson, Gerard and the Virgo Consortium. Halo and Galaxy Formation Histories from the Millennium Simulation, Public release of a VO-oriented and SQL-queryable database for studying the evolution of galaxies in the ΛCDM cosmogony, http://www.mpa-garching.mpg.de/millennium/. [cited at p. 6]

[Max05] Max-Planck-Gesellschaft. Supercomputer Simulations Explain the Formation of Galaxies and Quasars in the Universe, http://www.mpg.de/512207/pressRelease20050517, June 2005. [cited at p. 6]

[MB95] C.-P. Ma and E. Bertschinger. Cosmological Perturbation Theory in the Synchronous and Conformal Newtonian Gauges. *apj*, 455:7–+, December 1995. [cited at p. 28]

[Mil09] Iulian Constantin Milaş. N-Body problem, simulating the formation and evolution of star clusters with collisions. Thesis, University Politehnica of Bucharest, Spl. Independenţei 313, Bucharest 060042, Romania, June 2009. [cited at p. 2, 4, 9]

[NBo] NBodyLab.org. GRAPE Special Purpose Computers, http://www.nbodylab.com/nbl_grape_history.html. [cited at p. 5]

[Pro11] Programul Operational Sectorial Cresterea Competitivitatii Economice, cofinantat prin Fondul European de Dezvoltare Regionala Investitii pentru viitorul dumneavoastra, http://cluster.grid.pub.ro/. *The NCIT Cluster Resources User's Guide*, April 2011. [cited at p. 20]

[Sch07] Jakub Schwarzmeier. How to Start With Galaxy Dynamics Computer Simulations, http://www.kof.zcu.cz/st/dis/schwarzmeier/. Manual, University of West Bohemia in Pilsen, Department of General Physics, April 2007. [cited at p. 10]

[Spi11] Victor-Lucian Spiridon. Videos of the N-Body Simulations With GADGET-2, http://swarm.cs.pub.ro/~victor.spiridon/gadget2/, June 2011. [cited at p. 31, 32, 33, 35]

[Spr05a] Volker Springel. The cosmological simulation code GADGET-2. Paper, Max-Planck-Institute for Astrophysics, Karl-Schwarzschild-Straβe 1, 85740 Garching bei Munchen, Germany, May 2005. [cited at p. 5, 10, 14, 15, 16]

[Spr05b] Volker Springel. User guide for GADGET-2. Manual, Max-Plank-Institute for Astrophysics, Garching, Germany, May 2005. [cited at p. 23, 24, 26, 56, 61]

[Sun07] Sun Microsystems, Inc. *Sun N1 Grid Engine 6.1 User's Guide*, May 2007. [cited at p. 21]

[Sun08] Sun Microsystems, Inc. *Beginner's Guite to Sun Grid Engine 6.2*, September 2008. [cited at p. 21]

[SWTK01] V. Springel, S.D.M. Whide, G. Tormen, and G. Kauffmann. Populating a cluster of galaxies, results at z=0. Paper, Mon. Not. R. Astron. Soc.,328, 726, 2001. [cited at p. 40]

[SYW00] Volker Springel, Naoki Yoshida, and Simon D.M. White. GADGET: a code for collisionless and gasdynamical cosmological simulations. Paper, Max-Planck-Institute for Astrophysics, Karl-Schwarzschild-Straβe 1, 85740 Garching bei Munchen, Germany, December 2000. [cited at p. 10, 15]

[Tey02] Teyssier, R. Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES. *aap*, 385:337–364, April 2002. [cited at p. 5]

[Wet09] Wetzstein, M. and Nelson, A. F. and Naab, T. and Burkert, A. VINE – A Numerical Code for Simulating Astrophysical Systems Using Particles. I. Description of the Physics and the Numerical Methods. *apjs*, 184:298–325, October 2009. [cited at p. 5]

[Wri06] E. L. Wright. A Cosmology Calculator for the World Wide Web, http://www.astro.ucla.edu/~wright/CosmoCalc.html, December 2006. [cited at p. 19]

[Wri09] E. L. Wright. Ned Wright's Cosmology Tutorial, http://www.astro.ucla.edu/~wright/cosmo_01.htm, May 2009. [cited at p. 19]

Listings

Appendices

Appendix A – Performance Analysis

Time Results for the "Galaxy" simulation

	OpenMPI w/o OpenIB			OpenMPI with OpenIB			
N	T/step	Speedup	Efficiency	T/step	Speedup	Efficiency	Steps
1	1,02	1,00	1,00	1,01	1,00	1,00	512
2	0,62	1,65	0,82	0,63	1,61	0,80	512
4	0,38	2,66	0,67	0,38	2,68	0,67	512
6	0,30	3,38	0,56	0,30	3,34	0,56	512
8	0,25	4,11	0,51	0,25	4,10	0,51	512
12	0,20	5,19	0,43	0,20	5,15	0,43	512
24	0,34	3,01	0,13	0,19	5,42	0,23	512
36	0,46	2,19	0,06	0,24	4,24	0,12	512
48	0,66	1,55	0,03	0,35	2,86	0,06	512

Table 1: Time/step, Speedup and Efficiency for the "Galaxy" simulation.

Time Results for the "Cluster" simulation

	OpenMPI w/o OpenIB			OpenMPI with OpenIB			
N	T/step	Speedup	Efficiency	T/step	Speedup	Efficiency	Steps
1	5,65	1,00	1,00	5,68	1,00	1,00	417
2	3,16	1,79	0,89	3,17	1,79	0,90	417
4	1,72	3,28	0,82	1,72	3,29	0,82	417
6	1,26	4,47	0,74	1,26	4,50	0,75	417
8	1,01	5,62	0,70	1,00	5,67	0,71	417
12	0,77	7,37	0,61	0,75	7,53	0,63	417
24	1,14	4,96	0,21	0,51	11,20	0,47	417
36	1,45	3,89	0,11	0,46	12,48	0,35	417
48	1,42	3,97	0,08	0,48	11,90	0,25	417
60	1,80	3,14	0,05	0,58	9,81	0,16	417
72	1,91	2,95	0,04	0,71	8,06	0,11	417
84	2,32	2,44	0,03	1,00	5,68	0,07	417

Table 2: Time/step, Speedup and Efficiency for the "Cluster" simulation.

Time Results for the "Gas" simulation

	OpenMPI w/o OpenIB			OpenMPI with OpenIB			
N	T/step	Speedup	Efficiency	T/step	Speedup	Efficiency	Steps
1	0,71	1,00	1,00	0,70	1,00	1,00	602
2	0,36	1,96	0,98	0,38	1,84	0,92	602
4	0,23	3,13	0,78	0,22	3,08	0,77	606
6	0,17	4,23	0,71	0,17	4,14	0,69	608
8	0,14	5,04	0,63	0,14	4,92	0,62	607
12	0,11	6,59	0,55	0,11	6,31	0,53	603
24	0,25	2,86	0,12	0,10	6,98	0,29	607
36	0,40	1,78	0,05	0,14	5,15	0,14	601
48	0,60	1,18	0,02	0,17	4,06	0,08	600

Table 3: Time/step, Speedup and Efficiency for the "Gas" simulation.

Time Results for the "Density Fields" simulation

	Time per Step						Speedup						Efficiency					
N	128^3	192^3	256^3	384^3	512^3	768^3	128^3	192^3	256^3	384^3	512^3	768^3	128^3	192^3	256^3	384^3	512^3	768^3
1	58,49	186,14	463,99	1467,36	3711*	11738*	1,00	1,00	1,00	1,00	1*	1*	1,00	1,00	1,00	1,00	1,00	1,00
2	29,95	99,43	234,61	792,94	1876*	6343*	1,95	1,87	1,98	1,85	2*	2*	0,98	0,94	0,99	0,93	0,99	0,93
4	15,94	51,74	126,79	408,83	1014*	3270*	3,67	3,60	3,66	3,59	4*	3*	0,92	0,90	0,91	0,90	0,91	0,90
6	10,92	35,96	87,49	287,24	699*	2297*	5,36	5,18	5,30	5,11	5*	5*	0,89	0,86	0,88	0,85	0,88	0,85
8	8,23	27,19	63,78	217,10	510*	1736*	7,11	6,85	7,28	6,76	7*	6*	0,89	0,86	0,91	0,84	0,91	0,84
10	6,98	23,16	55,97	182,43	447*	1459*	8,38	8,04	8,29	8,04	8*	8*	0,84	0,80	0,83	0,80	0,83	0,80
12	5,70	19,62	46,77	156,73	374*	1253*	10,26	9,49	9,92	9,36	10*	10*	0,85	0,79	0,83	0,78	0,83	0,78
24	3,50	10,35	25,57	79,36	223,21	634*	16,71	17,99	18,15	18,49	16,63	18*	0,7	0,75	0,76	0,77	0,69	0,77
36	2,15	7,60	17,85	56,85	138,50	454*	27,16	24,49	25,99	25,81	26,80	25*	0,75	0,68	0,72	0,72	0,74	0,72
48	2,95	5,42	14,59	44,50	107,31	356*	19,85	34,32	31,80	32,97	34,59	32*	0,41	0,72	0,66	0,69	0,72	0,69
60	1,68	5,37	12,30	36,33	86,67	290*	34,82	34,68	37,71	40,39	42,83	40*	0,58	0,58	0,63	0,67	0,71	0,67
72	1,25	3,76	10,53	31,75	76,00	254*	46,79	49,46	44,08	46,21	48,84	46*	0,65	0,69	0,61	0,64	0,68	0,64
84	1,71	3,89	8,54	28,10	67,47	322,48	34,20	47,81	54,33	52,21	55,01	36,4	0,41	0,57	0,65	0,62	0,65	0,43
96	0,93	4,29	10,24	25,47	59,67	200,75	62,89	43,42	45,30	57,61	62,21	58,47	0,66	0,45	0,47	0,60	0,65	0,61
108	1,45	2,72	8,25	22,91	51,56	167,37	40,43	68,52	56,26	64,05	72,00	70,14	0,37	0,63	0,52	0,59	0,67	0,65
120	1,51	3,23	7,53	24,22	48,91	153,81	38,82	57,63	61,65	60,59	75,89	76,32	0,32	0,48	0,51	0,50	0,63	0,64
132		2,78	5,65	20,35	47,97	142,76		67,04	82,07	72,09	77,38	82,23		0,51	0,62	0,55	0,59	0,62
Memory Consumption:							0,52G	1,77G	4,2G	13,8G	32,8G	111G	+ 100MB / processor (comm. buffer)					

Table 4: Time/step, Speedup and Efficiency for the "Density Fields" simulation for sizes of the problem from 128^3 to 768^3. The values marked with * are approximated by multiplying with 8 the values from 256^3 and 384^3, in order to get the Speedup.

Processing Speed of Gadget-2 on the opteron queue

	Particles/Second										
N	Galaxy	Cluster	Gas	128^3	192^3	208^3	256^3	384^3	416^3	512^3	768^3
1	59.406	48.679	93.623	35.855	38.025	36.617	36.159	38.588	36.468		
2	95.238	87.223	172.463	70.022	71.185	69.916	71.511	71.409	70.991		
4	157.895	160.755	297.891	131.565	136.797	133.337	132.323	138.500	135.329		
6	200.000	219.443	385.506	192.047	196.827	194.403	191.762	197.128	195.061		
8	240.000	276.498	468.114	254.818	260.312	261.596	263.048	260.816	267.020		
10				300.452	305.608	310.736	299.754	310.383	303.019		
12	300.000	368.664	595.782	367.921	360.749	357.241	358.717	361.278	317.997		
24	315.789	542.153	655.360	599.186	683.854	705.244	656.129	713.497	698.131	601.307	
36	250.000	601.083	468.114	975.420	931.301	919.194	939.900	996.009	978.541	969.081	
48	171.429	576.038	385.506	710.899	1.305.883	1.294.807	1.149.912	1.272.429	1.197.261	1.250.748	
60		476.721		1.248.305	1.318.042	1.187.192	1.364.001	1.558.577	1.578.410	1.548.607	
72		389.434		1.677.722	1.882.417	1.426.135	1.593.278	1.783.405	1.693.116	1.766.023	
84		276.498		1.226.405	1.819.508	1.832.772	1.964.545	2.015.057	1.909.077	1.989.295	1.404.691
96				2.255.002	1.649.857	1.778.441	1.638.400	2.223.129	2.124.264	2.249.333	2.256.462
108				1.446.312	2.602.165	2.386.979	2.033.602	2.471.545	2.347.287	2.603.137	2.706.488
120				1.388.842	2.191.297	1.743.975	2.228.050	2.337.866	2.526.897	2.744.178	2.945.093
132					2.546.003	1.298.544	2.969.419	2.782.462	2.736.271	2.797.951	3.173.051

Table 5: Processing speed of GADGET-2 on the opteron queue for different simulations.

	Particles/Second per Processor										
N	Galaxy	Cluster	Gas	128^3	192^3	208^3	256^3	384^3	416^3	512^3	768^3
1	59.406	48.679	93.623	35.855	38.025	36.617	36.159	38.588	36.468		
2	47.619	43.612	86.232	35.011	35.592	34.958	35.756	35.705	35.496		
4	39.474	40.189	74.473	32.891	34.199	33.334	33.081	34.625	33.832		
6	33.333	36.574	64.251	32.008	32.804	32.400	31.960	32.855	32.510		
8	30.000	34.562	58.514	31.852	32.539	32.700	32.881	32.602	33.378		
10				30.045	30.561	31.074	29.975	31.038	30.302		
12	25.000	30.722	49.648	30.660	30.062	29.770	29.893	30.107	26.500		
24	13.158	22.590	27.307	24.966	28.494	29.385	27.339	29.729	29.089	25.054	
36	6.944	16.697	13.003	27.095	25.869	25.533	26.108	27.667	27.182	26.919	
48	3.571	12.001	8.031	14.810	27.206	26.975	23.956	26.509	24.943	26.057	
60		7.945		20.805	21.967	19.787	22.733	25.976	26.307	25.810	
72		5.409		23.302	26.145	19.807	22.129	24.770	23.516	24.528	
84		3.292		14.600	21.661	21.819	23.387	23.989	22.727	23.682	16.723
96				23.490	17.186	18.525	17.067	23.158	22.128	23.431	23.505
108				13.392	24.094	22.102	18.830	22.885	21.734	24.103	25.060
120				11.574	18.261	14.533	18.567	19.482	21.057	22.868	24.542
132					19.288	9.837	22.496	21.079	20.729	21.197	24.038

Table 6: Average processing speed per processor of GADGET-2 on the opteron queue for different simulations.

Note: The processing speed of the 768^3 run for 72 processors is lower than expected due to a very little amount of insufficient memory resulting in the use of swap.

Profiling

Profiling of GADGET-2 using Oracle Solaris Studio 12.2
Note that the code runs slower because of the profiling overhead.

The "Gas" Simulation, 600 steps

Figure 1: Profiling for the "Gas" simulation. The System CPU time is associated with MPI communication overhead over TCP/IP.

The "Density Fields" Simulation, N = 256^3, 4 steps

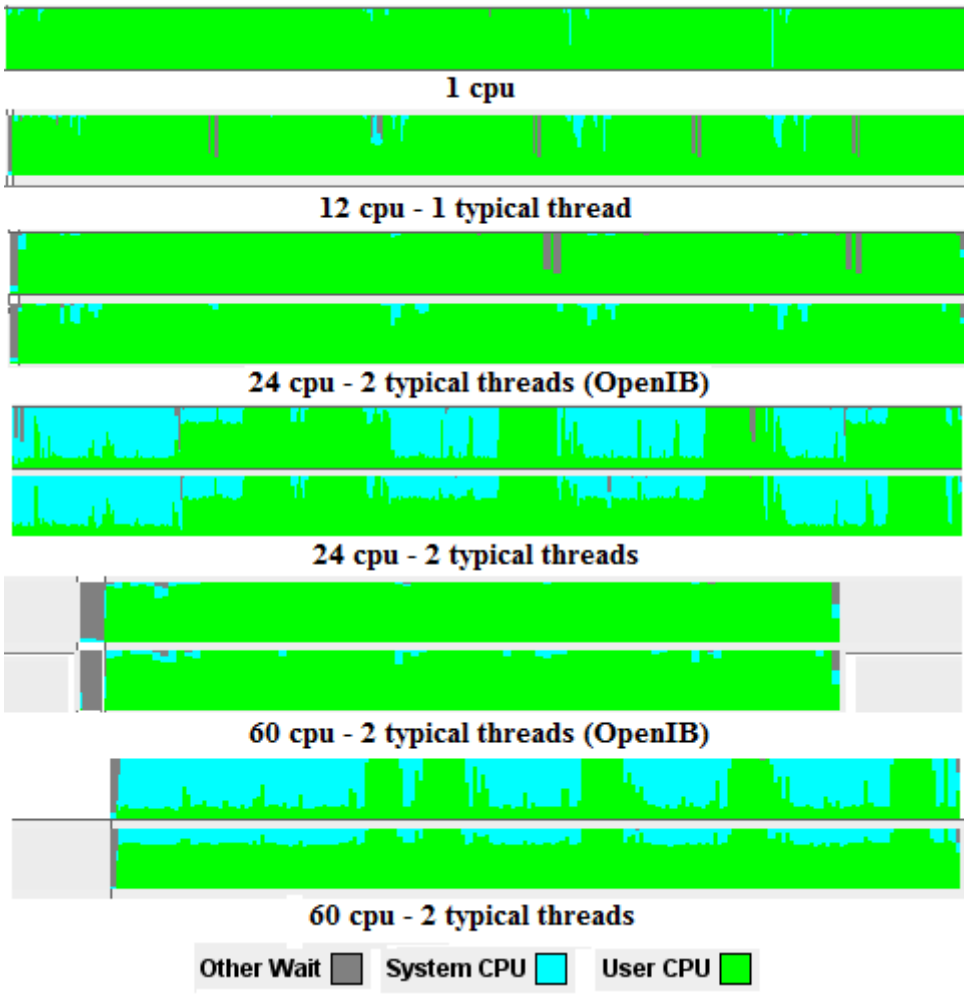

Figure 2: Profiling for the "Density Fields" simulation, N = 256^3. The System CPU time is associated with MPI communication overhead over TCP/IP.

Type	Duration	User	System &Other	MPI	Hydro force	Gravity tree	Density	Long-range force	Domain decomp.	Find timestep
1 CPU	733s	99,48%	0,52%	1,4s (0.19%)	232s (32%)	208s (28%)	138s (19%)	96s (13%)	24s (3%)	13s (1%)
12 CPU	1.488s	98,09%	1,91%	21s (1,41%)	496s (34%)	376s (26%)	243s (17%)	192s (13%)	83s (6%)	25s (2%)
24 CPU IB	2.556s	96,94%	3,06%	1.337s (52%)	697s (28%)	563s (23%)	385s (16%)	273s (11%)	322s (13%)	57s (2%)
24 CPU	3.970s	60,89%	39,11%	2.708s (68%)	786s (20%)	606s (16%)	428s (11%)	835s (22%)	364s (9%)	154s (4%)
36 CPU IB	4.805s	94,41%	5,59%	2.940s (61%)	1.180s (25%)	791s (17%)	489s (10%)	380s (8%)	1.306s (28%)	142s (3%)
36 CPU	9.940s	54,42%	45,58%	8.049s (81%)	1.238s (13%)	1.166s (12%)	879s (9%)	2.520s (26%)	1.692s (17%)	552s (6%)

Table 7: Time consumption of various portions of the code for the "Gas" simulation, 600 steps. Duration is summed over all CPUs.

Type	Duration	User	System &Other	MPI	Gravity tree	Long-range force	Domain decomp.	Find timestep
1 CPU	3.878s	98,83%	1,17%	2,8s (0,07%)	3.408s (88%)	225s (6%)	118s (3%)	42s (1%)
12 CPU	4.533s	97,43%	2,57%	470s (10,3%)	3.587s (80%)	482s (11%)	171s (4%)	67s (2%)
24 CPU IB	4.792s	98,08%	1,92%	710s (14,8%)	3.702s (79%)	506s (11%)	177s (4%)	62s (2%)
24 CPU	9.882s	67,90%	32,10%	5.684s (58%)	4.504s (42%)	3.612s (37%)	198s (2%)	86s (1%)
60 CPU IB	5.799s	97,84%	2,16%	1.503s (26%)	4.143s (74%)	652s (12%)	230s (4%)	64s (2%)
60 CPU	12.259s	65,28%	34,72%	7.927s (65%)	4.656s (39%)	4.133s (34%)	189s (2%)	66s (1%)

Table 8: Time consumption of various portions of the code for the "Density Fields" simulation, N = 256^3, 4 steps. Duration is summed over all CPUs.

Appendix B – Hardware and Software Configurations

NCIT Cluster Blade Hardware Specifications

The hardware specifications for the quad, opteron and nehalem nodes.

No. of Processors	2
No. of Physical Cores	6 per processor
Physical Memory	16 GByte
Processor Name	AMD Opteron 2435
Frequency	2.6 GHz
Process	45 nm
TDP	75 W
L1 cache	128 KB per core
L2 cache	512 KB per core
L3 cache	6 MB accessible by all cores
Memory Type	Dual-channel ECC registered DDR2 - PC2-6400 (800 Mhz)
cpubenchmark.net score	9840 (dual CPU)

Table 9: IBM BladeCenter LS22 Specifications.

No. of Processors	2
No. of Physical Cores	4 per processor
Physical Memory	16 GByte
Processor Name	Intel Xeon E5405
Frequency	2.0 GHz
Process	45 nm
TDP	80 W
L1 cache	64 KB per core
L2 cache	2x6 MB per core
FSB	1333 MT/s
Memory Type	Dual-Channel DDR3 - PC3-8500 (1033 Mhz)
cpubenchmark.net score	6128 (dual CPU)

Table 10: IBM BladeCenter HS21 Specifications.

No. of Processors	2
No. of Physical Cores	4 (8 with HT) per processor
Physical Memory	32 GByte
Processor Name	Intel Xeon E5630
Frequency	2.53 GHz
Process	32 nm
TDP	80 W
L1 cache	64 KB per core
L2 cache	256 KB per core
L3 cache	12 MB accessible by all cores
I/O Bus	3 x 5.86 GT/s QPI
Memory Type	Triple-channel DDR3 - PC3-8500 (1033 Mhz)
cpubenchmark.net score	9469 (dual CPU)

Table 11: IBM BladeCenter HS22 Specifications.

GADGET-2 Dependencies

Below is a table containing the packages and versions installed in order to use GADGET-2

Package	Dependencies
gcc-4.6.0	cloog-ppl-0.15.11
	ppl-0.11.2
	mpc-0.8.1
	gmp-4.3.2
	mpfr-2.4.2
openmpi-1.5.3	gcc-4.6.0
openmpi-1.5.3	gcc-4.6.0
with OpenIB	libibverbs-devel-1.1.2-1.el5.i386.rpm
	libibverbs-devel-1.1.2-1.el5.x86_64.rpm
	libibverbs-static-1.1.2-1.el5.x86_64.rpm
	libibverbs-utils-1.1.2-1.el5.x86_64.rpm
	libsysfs-2.0.0-6.i386.rpm
	libsysfs-devel-2.0.0-6.i386.rpm
	libsysfs-devel-2.0.0-6.x86_64.rpm
GADGET-2	openmpi-1.5.3
	fftw-2.1.5
	gsl-1.13
	hdf5-1.6.10
	szip-2.1

Table 12: List of installed packages.

GADGET-2 Configuration Files

Example of the Makefile [1] and parameter file [2] for a simulation with 256^3 particles in a periodic box with comoving integration. For a full explanation of the options and parameters see [Spr05b].

Listing 1: GADGET-2 Makefile options.

```
1
2 #--------------- Basic operation mode of code
3 OPT   += -DPERIODIC
4 #OPT   += -DUNEQUALSOFTENINGS
5 #--------------- Things that are always recommended
6 OPT   += -DPEANOHILBERT
7 OPT   += -DWALLCLOCK
8 #--------------- TreePM Options
9 OPT   += -DPMGRID=512
```

```
10 #OPT    +=  -DPLACEHIGHRESREGION=3
11 #OPT    +=  -DENLARGEREGION=1.2
12 #OPT    +=  -DASMTH=1.25
13 #OPT    +=  -DRCUT=4.5
14 #--------------- Single/Double Precision
15 #OPT    +=  -DDOUBLEPRECISION
16 #OPT    +=  -DDOUBLEPRECISION_FFTW
17 #--------------- Time integration options
18 OPT    +=  -DSYNCHRONIZATION
19 #OPT    +=  -DFLEXSTEPS
20 #OPT    +=  -DPSEUDOSYMMETRIC
21 OPT    +=  -DNOSTOP_WHEN_BELOW_MINTIMESTEP
22 #OPT    +=  -DNOPMSTEPADJUSTMENT
23 #--------------- Output options
24 #OPT    +=  -DHAVE_HDF5
25 #OPT    +=  -DOUTPUTPOTENTIAL
26 #OPT    +=  -DOUTPUTACCELERATION
27 #OPT    +=  -DOUTPUTCHANGEOFENTROPY
28 #OPT    +=  -DOUTPUTTIMESTEP
29 #--------------- Things for special behaviour
30 #OPT    +=  -DNOGRAVITY
31 #OPT    +=  -DNOTREERND
32 #OPT    +=  -DNOTYPEPREFIX_FFTW
33 #OPT    +=  -DLONG_X=60
34 #OPT    +=  -DLONG_Y=5
35 #OPT    +=  -DLONG_Z=0.2
36 #OPT    +=  -DTWODIMS
37 #OPT    +=  -DSPH_BND_PARTICLES
38 #OPT    +=  -DNOVISCOSITYLIMITER
39 #OPT    +=  -DCOMPUTE_POTENTIAL_ENERGY
40 #OPT    +=  -DLONGIDS
41 #OPT    +=  -DISOTHERMAL
42 #OPT    +=  -DSELECTIVE_NO_GRAVITY=2+4+8+16
43 #--------------- Testing and Debugging options
44 #OPT    +=  -DFORCETEST=0.1
45 #--------------- Glass making
46 #OPT    +=  -DMAKEGLASS=262144
```

Listing 2: GADGET-2 Parameter file options.

```
1 %  Relevant files
2 InitCondFile       ./small.gadget
3 OutputDir          ./
4 EnergyFile         energy.txt
5 InfoFile           info.txt
6 TimingsFile        timings.txt
7 CpuFile            cpu.txt
```

```
 8 RestartFile        restart
 9 SnapshotFileBase   snapshot
10 OutputListFilename ./outputs_lcdm_gas.list
11
12 % CPU time -limit
13 TimeLimitCPU      360000  % = 100 hours
14 ResubmitOn        0
15 ResubmitCommand   my-scriptfile
16
17 % Code options
18 ICFormat                 1
19 SnapFormat               1
20 ComovingIntegrationOn    1
21 TypeOfTimestepCriterion  0
22 OutputListOn             1
23 PeriodicBoundariesOn     1
24
25 %  Caracteristics of run
26 TimeBegin           0.300167
27 TimeMax             1.10
28 Omega0                0.3
29 OmegaLambda           0.7
30 OmegaBaryon           0.0
31 HubbleParam           0.7
32 BoxSize               50000.0
33
34 % Output frequency
35 TimeBetSnapshot        1.2
36 TimeOfFirstSnapshot    1.2
37 CpuTimeBetRestartFile     36000.0     ; here in seconds
38 TimeBetStatistics         0.001
39 NumFilesPerSnapshot       1
40 NumFilesWrittenInParallel 1
41
42 % Accuracy of time integration
43 ErrTolIntAccuracy      0.02
44 MaxRMSDisplacementFac  0.2
45 CourantFac             0.15
46 MaxSizeTimestep       0.001
47 MinSizeTimestep       0.001
48
49 % Tree algorithm, force accuracy, domain update frequency
50 ErrTolTheta            0.5
51 TypeOfOpeningCriterion 1
52 ErrTolForceAcc         0.005
53 TreeDomainUpdateFrequency    0.1
54
```

```
55 %  Further parameters of SPH
56 DesNumNgb              33
57 MaxNumNgbDeviation     2
58 ArtBulkViscConst       0.8
59 InitGasTemp            1000.0   % always ignored if set to 0
60 MinGasTemp             50.0
61
62 % Memory allocation
63 PartAllocFactor       1.4
64 TreeAllocFactor       0.8
65 BufferSize            100          % in MByte
66
67 % System of units
68 UnitLength_in_cm         3.085678e21   ;  1.0 kpc
69 UnitMass_in_g            1.989e43      ;  1.0e10 solar masses
70 UnitVelocity_in_cm_per_s 1e5           ;  1 km/sec
71 GravityConstantInternal  0
72
73 % Softening lengths
74 MinGasHsmlFractional 0.25
75 SofteningGas       0
76 SofteningHalo      5.0
77 SofteningDisk      0
78 SofteningBulge     0
79 SofteningStars     0
80 SofteningBndry     0
81 SofteningGasMaxPhys       0
82 SofteningHaloMaxPhys      5.0
83 SofteningDiskMaxPhys      0
84 SofteningBulgeMaxPhys     0
85 SofteningStarsMaxPhys     0
86 SofteningBndryMaxPhys     0
```

Example of a `.bashrc` file configuring the paths for the compiler, mpi and necessary libraries.

Listing 3: Configuring paths in the ***.bashrc*** file.

```
1
2 #gcc
3 TMP=opt/gcc-4.6.0
4 export LIBRARY_PATH=$HOME/$TMP/lib64:$LIBRARY_PATH
5 export LD_LIBRARY_PATH=$HOME/$TMP/lib64:$LD_LIBRARY_PATH
6 export PATH=$HOME/$TMP/bin:$PATH
7
8 #mpi
9 OPENIB=_openib
10 TMP=opt/openmpi-1.5.3_gcc-4.6.0$OPENIB
```

```
11 export LIBRARY_PATH=$HOME/$TMP/lib:$LIBRARY_PATH
12 export LD_LIBRARY_PATH=$HOME/$TMP/lib:$LD_LIBRARY_PATH
13 export PATH=$HOME/$TMP/bin:$PATH
14
15 #dependencies
16 export LD_LIBRARY_PATH=$HOME/opt/lib:$LIBRARY_PATH
17 export LIBRARY_PATH=$HOME/opt/lib:$LD_LIBRARY_PATH
18 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH/usr/lib64:/lib64
19 export LIBRARY_PATH=$LIBRARY_PATH/usr/lib64:/lib64
```

Running GADGET-2 on the NCIT cluster

Listing 4: Status of the working nodes, ***./free_nodes.sh opteron*** sample output.

```
1
2  queuename                qtype      resv/used/tot  load_avg      arch
3  ibm-opteron.q@opteron-wn01.gri BP    0/4/12     0.09    lx24-amd64
4  ibm-opteron.q@opteron-wn02.gri BP    0/0/12     0.14    lx24-amd64
5  ibm-opteron.q@opteron-wn03.gri BP    0/0/12     0.05    lx24-amd64
6  ibm-opteron.q@opteron-wn04.gri BP    0/0/12     0.06    lx24-amd64
7  ibm-opteron.q@opteron-wn05.gri BP    0/0/12     0.03    lx24-amd64
8  ibm-opteron.q@opteron-wn06.gri BP    0/0/12     0.08    lx24-amd64
9  ibm-opteron.q@opteron-wn07.gri BP    0/0/12     0.09    lx24-amd64
10 ibm-opteron.q@opteron-wn08.gri BP    0/0/12     0.06    lx24-amd64
11 ibm-opteron.q@opteron-wn09.gri BP    0/0/12     0.05    lx24-amd64
12 ibm-opteron.q@opteron-wn10.gri BP    0/0/12     0.06    lx24-amd64
13 ibm-opteron.q@opteron-wn11.gri BP    0/0/12     0.94    lx24-amd64
14 ibm-opteron.q@opteron-wn12.gri BP    0/0/12     0.05    lx24-amd64
15 ibm-opteron.q@opteron-wn13.gri BP    0/0/12     0.06    lx24-amd64
16 ibm-opteron.q@opteron-wn14.gri BP    0/0/12     1.11    lx24-amd64
```

7.2 GADGET-2 Memory Consumption

Let N_{tot} be the total number of particles and N_p the number of processors. Then the average particle load per processor is $N = N_{tot}/N_p$. The code will allocate about:

$$M_1 = PartAllocFactor \times N \times (68 + TreeAllocFactor \times 64)$$

bytes for particle storage, for the gravity and neighbour tree, and for the time integration scheme, on each processor. For typical values like $TreeAllocFactor = 0,8$ and $PartAllocFactor = 1,4$, this implies a memory cost of ~ 167 bytes per particle. For SPH particles, additional memory of

$$M_2 = PartAllocFactor \times N_{sph} \times 84$$

bytes will be required. For an equal mixture of SPH and collisionless particles, the average memory consumption per partile is therefore going to be about 225 bytes per particle.

Note however that it is not advisable to fill all the available memory of the machine. For the sake of performance one should leave room for memory-imbalance, i.e. $PartAllocFactor$ needs to be chosen larger than 1,0. In practice, values below 142 can lead to serious work-imbalance, depending on the amount of clustering and on how many processors are involved.

Certain code options will also increase the amount of memory allocated per particle. Obviously, selecting `DOUBLEPRECISION` can increase the storage requirement substantially, almost doubling it.

In practice, it is thus best to pay attention to the screen output that is produced in the log-file when GADGET-2 starts. The code will report the sizes of all major chunks of memory that are allocated. Note that these numbers refer to the allocation done by each individual processor. To obtain the total amount of memory needed by the code, one needs to multiply the sum of these numbers by the number of processors that are used.

The communication buffers requires:

$$M_3 = BufferSize \times 1024^2$$

bytes, but this number is independent of the particle load.

Note that the TreePM algorithm can require substantial amounts of additional memory, depending on the choice of `PMGRID`, and the type of TreePM simulation done. In the simplest case, a periodic box, the allocated amount of memory, summed over all processors, is about

$$M_4 = PMGRID^3 \times 12 - 16$$

bytes, provided single precision is used. For double precision, twice this amount is needed. If the mesh is made a factor of 2 finer than the mean particle spacing in each dimension, then the required storage for the PM algorithm roughly matches that needed for the particle data and the tree, i.e. then the memory requirement of the code essentially doubles.

If the TreePM algorithm with non-periodic boundaries is selected instead, the memory requirements are substantially larger. They are then roughly:

$$M_5 = (2 \times PMGRID)^3 \times 16$$

bytes. If a two level zoom simulation in a periodic volume is selected, then M_4 and M_5 are both allocated. If the zoom is done in a simulation with vacuum boundaries, then only M_5 is allocated, but the number 16 is replaced by 20. [Spr05b]

Diagnostic output files

The numbers give measurements for the following parts (in that sequence) in the CpuFile:

1	Total consumption
2	Gravitational force computation (including tree walks, construction, communication)
3	Hydrodynamics (including neighbour search, density determination, and SPH communication)
4	Domain decomposition
5	Potential energy computation (for the energy statistics)
6	Drifting the particle set
7	Timestep determination and kicking of particles
8	Writing of snapshot files
9	Pure time for tree walks (this is a sub-part of 1)
10	Tree construction (this is a sub-part of 1)
11	Communication and summation (this tries to measure the time spent in communication and summation of force components in the gravitational tree part)
12	Imbalance of gravitational tree (this measures work-load imbalance losses directly)
13	SPH computations (finding of neighbours and evaluating SPH-sums with them)
14	Communication and summation in the SPH routines
15	Losses due to work-load imbalance in the SPH part
16	Neighbour adjustment (measures the time needed to readjust SPH smoothing lengths if the number of neighbours obtained for the next guess smoothing length falls outside the desired range)
17	Time for PM computation
18	Time needed to establish Peano-Hilbert order

Table 13: Time measurements in the CpuFile.

Writing initial conditions

The default gadget file format:

See the table on the next page:

No.	Details
1.	**Header: struct SnapshotHeader** The individual fields of the file header are defined in Table 15
2.	**Particle positions: float pos[N][3]** Comoving coordinates in internal length units (corresponds to h^1kpc if the above choice for the system of units is adopted). `N=sum(SnapshotHeader.Npart)` is the total number of particles in the file. Note: The particles are ordered in the file according to their type, so it is guaranteed that particles of type 0 come before those of type 1, and so on. However, within a type, the sequence of particles can change from dump to dump.
3.	**Particle velocities: float vel[N][3]** Particle velocities u in internal velocity units (corresponds to km/sec if the default choice for the system of units is adopted). Peculiar velocities v are obtained by multiplying u with $\sqrt{a}$, i.e. $v = u\sqrt{a}$. For non-cosmological integrations, $u = v$ are just ordinary velocities.
4.	**Particle identifications: unsigned int id[N]** Particle number for unique identification. The order of particles may change between different output dumps, but the IDs can be easily used to bring the particles back into the original order for every dump.
5.	**Variable particle masses: float masses[Nm]** Particle masses for those particle types that have variable particle masses (i.e. zero entries in `SnapshotHeader.massarr`). Nm is the sum of those npart entries that have vanishing `massarr`. If `Nm=0`, this block is not present at all.
6.	**Internal energy: float u[Ngas]** Internal energy per unit mass for the `Ngas=npart(0)` gas particles in the file. The block is only present for $Ngas > 0$. Units are again in internal code units, i.e. for the above system of units, u is given in $(\mathrm{km/sec})^2$.
7.	**Density: float rho[Ngas]** Comoving density of SPH particles. Units are again in internal code units, i.e. for the above system of units, `rho` is given in $10^{10}h^{-1}M_\odot/(h^{-1}kpc)^3$.
8.	**Smoothing length: float hsml[Ngas]** Smoothing length of the SPH particles. Given in comoving coordinates in internal length units.
9.	**Gravitational potential: float pot[N]** Gravitational potential for particles.
10.	**Accelerations: float acc[N][3]** Accelerations of particles.
11.	**Rate of entropy production: float dAdt[Ngas]** The rate of change of the entropic function of each gas particles. For adiabatic simulations, the entropy can only increase due to the implemented viscosity.
12.	**Timesteps of particles: float dt[N]** Individual particle timesteps of particles. For cosmological simulations, the values stored here are $\Delta ln(a)$, i.e. intervals of the natural logarithm of the expansion factor.

Table 14: Blocks in the GADGET standard file format.

Fields in the SnapshotHeader data structure

Header Field	Type	Comment
Npart[6]	uint	The number of particles of each type in the present file.
Massarr[6]	double	The mass of each particle type. If set to 0 for a type which is present, individual particle masses from the `mass` block are read instead.
Time	double	Time of output, or expansion factor for cosmological simulations.
Redshift	double	$z = 1/a1$ (only set for cosmological integrations).
FlagSfr	int	Flag for star formation (unused in the public version of GADGET-2).
FlagFeedback	int	Flag for feedback (unused).
Nall	int	Total number of particles of each type in the simulation.
FlagCooling	int	Flag for cooling (unused).
Numfiles	int	Number of files in each snapshot.
BoxSize	double	Gives the box size if periodic boundary conditions are used.
Omega0	double	Matter density at $z = 0$ in units of the critical density (only relevant for cosmological integrations).
OmegaLambda	double	Vacuum energy density at $z = 0$ in units of the critical density.
HubbleParam	double	Gives h, the Hubble constant in units of $100 kms^{-1} Mpc^1$ (for cosmological integrations).
FlagAge	int	Creation times of stars (unused).
FlagMetals	int	Flag for metallicity values (unused).
NallHW[6]	int	For simulations that use more than 2^{32} particles, these fields hold the most significant word of 64-bit total particle numbers. Otherwise zero.
Flag_entr_ics	int	Flags that *initial conditions* contain entropy instead of thermal energy in the `u` block.
Unused		Currently unused space which fills the header to a **total length of 256 bytes**, leaving room for future additions.

Table 15: Fields in the SnapshotHeader structure.

Example of a program for writing initial conditions

Listing 5: Example of a program for writing initial conditions.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define  PI            3.14159265358979323846
#define  GRAVITY       6.672e-8
#define  HUBBLE        3.2407789e-18   /* in h/sec */

typedef unsigned int uint;

double UnitLength_in_cm         = 3.085678e21;
double UnitMass_in_g            = 1.989e43;
double UnitVelocity_in_cm_per_s = 1e5;

double G;
double UnitTime_in_s;
double Hubble;

struct SnapshotHeader{
    uint    npart[6];
    double  mass[6];
    double  time;
    double  redshift;
    int     flag_sfr;         // unused in public
    int     flag_feedback;    // unused in public
    uint    npartTotal[6];
    int     flag_cooling;     // radiative cooling
    uint    num_files;
    double  BoxSize;
    double  Omega0;           // matter density at z=0
    double  OmegaLambda;      // vacuum energy density at z=0
    double  HubbleParam;
    int     StellarAge;       // unused in public
    int     Metals;           // unused in public
    int     nAllHW[6];        // For than 2^32 particles. Otherwise zero.
    int     flag_entropy_ics;
    char    fill[60];  /* fills to 256 Bytes */
};

void set_units(void){        /* ... set some units */
    UnitTime_in_s = UnitLength_in_cm / UnitVelocity_in_cm_per_s;
```

```
44      G = GRAVITY / pow(UnitLength_in_cm, 3);
45      G *= UnitMass_in_g * pow(UnitTime_in_s, 2);
46      Hubble = HUBBLE * UnitTime_in_s;
47      }
48
49  double getCriticalRho() {
50      return 3 * Hubble * Hubble / (8 * PI * G);
51      }
52
53  int main( int argc,  char **argv ){
54      double BOXSIZE;
55      double CUBESIZE;
56      char *output_file = (char*)malloc(100);
57      uint N, N3root;
58      uint i, j, k;
59      uint ptype;
60      uint random_pos = 0;
61      uint random_vel = 0;
62      float temp_value;
63      float step;
64
65      printf("\nPlease enter Initial Conditions Variables:\n");
66      printf("CubicRoot(N_Particles) = ");
67      scanf("%u", &N3root);
68      printf("Box_Size = ");
69      scanf("%lf", &BOXSIZE);
70      printf("Cube_Size = ");
71      scanf("%lf", &CUBESIZE);
72      while( CUBESIZE > BOXSIZE ){
73          printf("Please enter Cube_Size __smaller__ than Box_Size\n");
74          printf("Cube_Size = "); scanf("%lf", &CUBESIZE);
75          }
76      printf("Particle Types: 0 = Gas; 1 = Halo; 2 = Disk; ");
77      printf("3 = Bulge; 4 = Stars; 5 = Boundry\nType = ");
78      scanf("%u", &ptype);
79      printf("Random positions (0|1) = ");
80      scanf("%u", &random_pos);
81      printf("Random velocities (0|1) = ");
82      scanf("%u", &random_vel);
83      printf("Output_file = ");
84      scanf("%s", output_file);
85
86      N = N3root * N3root * N3root;
87
88      struct SnapshotHeader head;
89      memset((void *)&head, 0, sizeof(struct SnapshotHeader));
90      head.npartTotal[ptype] = head.npart[ptype] = N;
```

```
91      head.redshift = 49.;
92      head.time = 1./(1. + head.redshift);
93      head.BoxSize = BOXSIZE;
94      head.Omega0 = 0.3;
95      head.OmegaLambda = 0.7;
96      head.HubbleParam = 0.7;
97
98      set_units();
99      double rho = (head.Omega0 - head.OmegaBaryon) * getCriticalRho();
100     double mass = rho * pow(head.BoxSize,3) / (double)N;
101     head.mass[ptype] = mass;
102     head.num_files = 1;
103     //everything else is set to 0 (zero)
104
105     FILE *f = fopen(output_file, "w");
106     uint blkSize = sizeof(struct SnapshotHeader);
107 // pad and write header
108     fwrite(&blkSize, sizeof(int), 1, f);
109     fwrite(&head, sizeof(struct SnapshotHeader), 1, f);
110     fwrite(&blkSize, sizeof(int), 1, f);
111
112     if (random_pos == 0) {
113         float values[N3root];
114         step = CUBESIZE / N3root;
115         //particles in the middle of the Box
116         temp_value = (BOXSIZE - CUBESIZE)/2  + step / N3root;
117         printf("Generated values: \n");
118         for (i = 0; i < N3root; i++){
119             values[i] = temp_value;
120             printf("%f, ", temp_value);
121             temp_value += step;
122             }
123     // pad and write positions
124         blkSize = N * 3 * sizeof(float);
125         fwrite(&blkSize, sizeof(int), 1, f);
126         for(i = 0; i < N3root; i++){
127             printf("%d ", i); fflush(stdout);
128             for(j = 0; j < N3root; j++)
129                 for(k = 0; k < N3root; k++){
130                     fwrite((void*)&values[i], sizeof(float), 1, f);
131                     fwrite((void*)&values[j], sizeof(float), 1, f);
132                     fwrite((void*)&values[k], sizeof(float), 1, f);
133                 }
134             }
135         fwrite(&blkSize, sizeof(int), 1, f);
136         printf("\npositions done\n");
137     }
```

```
138     else{ //random positions
139         srand(time(NULL));
140         // pad and write positions
141         blkSize = N * 3 * sizeof(float);
142         fwrite(&blkSize, sizeof(int), 1, f);
143         for(i = 0; i < N3root; i++){
144             printf("%d ", i); fflush(stdout);
145             for(j = 0; j < N3root; j++)
146                 for(k = 0; k < N3root; k++){
147         temp_value = ((float)(rand() % ((int)CUBESIZE*100)))/100.0;
148         fwrite((void*)&temp_value, sizeof(float), 1, f);
149         temp_value = ((float)(rand() % ((int)CUBESIZE*100)))/100.0;
150         fwrite((void*)&temp_value, sizeof(float), 1, f);
151         temp_value = ((float)(rand() % ((int)CUBESIZE*100)))/100.0;
152         fwrite((void*)&temp_value, sizeof(float), 1, f);
153                     }
154             }
155         fwrite(&blkSize, sizeof(int), 1, f);
156         printf("\npositions done\n");
157     }
158 // pad and write velocities
159     fwrite(&blkSize, sizeof(int), 1, f);
160     if (random_vel == 0){
161         temp_value = 0.0;
162         for(i = 0; i < 3*N; i++ ){
163             if(i % (3*N3root*N3root) == 0){
164                 printf("%d ", i/(3*N3root*N3root));
165                 fflush(stdout);
166                 }
167             fwrite((void*)&temp_value, sizeof(float), 1, f);
168         }
169     }
170     else{ //random velocities
171         for(i = 0; i < 3*N; i++ ){
172             if(i % (3*N3root*N3root) == 0){
173                 printf("%d ", i/(3*N3root*N3root));
174                 fflush(stdout);
175                 }
176             temp_value = (((float)(rand() % 200000))/100.0) - 1000.0;
177             fwrite((void*)&temp_value, sizeof(float), 1, f);
178             }
179     }
180     fwrite(&blkSize, sizeof(int), 1, f);
181     printf("velocities done\n");
182
183  // pad and write p-ID       0 -> N-1
184     blkSize = N * sizeof(uint);
```

```
185     fwrite(&blkSize, sizeof(int), 1, f);
186     for(i = 0; i < N; i++ ){
187         if(i % (N3root * N3root) == 0){
188                         printf("%d ", i/(N3root * N3root));
189                         fflush(stdout);
190                         }
191                 fwrite((void*)&i, sizeof(uint), 1, f);
192         }
193     fwrite(&blkSize, sizeof(int), 1, f);
194     printf("ids done\n");
195     fclose(f);
196     return 0;
197 }
```

Script for renaming snapshot files

By default GADGET-2 saves the name of the snapshots(configurable in the parameter file) according to the following rule: <**SnapshotFileBase**>_**yxxx** where **xxx** is a number from **000** to **999**. If there are more than 1000 files, then an additional digit is used. In order to rename the snapshots from _000 to _999 to _0000.bin to _0999.bin so that IFrIT can *animate* the files, we can use the following script:

Listing 6: Script for renaming the snapshots for use with IFrIT.

```
1 #The script needs to be run in the same folder
2 #where the files are
3 rename snapshot_ snapshot_0 snapshot_*
4 mmv -m "*" "#1.bin"
```