Scalable Parallel Simulation of Small-Scale Structure in Cold Dark Matter

by

Alexander V. Shirokov

B.S. Applied Mathematics and Physics, M.S. Theoretical Physics
Moscow Institute of Physics and Technology

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

March 2005 [June 2005]

Author .....
Department of Physics
March 21, 2005

Certified by .....
Edmund Bertschinger
Professor
Thesis Supervisor

Accepted by .....
Thomas Greytak
Professor of Physics
Associate Department Head for Education

Scalable Parallel Simulation of Small-Scale Structure in Cold Dark Matter

by

Alexander V. Shirokov

Submitted to the Department of Physics
on March 21, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We present a parallel implementation of the particle-particle/particle-mesh (P^3M) algorithm for distributed memory clusters. The `l1p3m-hc` code uses a hybrid method for both computation and domain decomposition. Long-range forces are computed using a Fourier transform gravity solver on a regular mesh; the mesh is distributed across parallel processes using a static one-dimensional slab domain decomposition. Short-range forces are computed by direct summation of close pairs; particles are distributed using a dynamic domain decomposition based on a space-filling Hilbert curve. A nearly-optimal method was devised to dynamically repartition the particle distribution so as to maintain load balance even for extremely inhomogeneous mass distributions. Tests using 800^3 simulations on a 40-processor Beowulf cluster showed good load balance and scalability up to 80 processes. We discuss the limits on scalability imposed by communication and extreme clustering and suggest how they may be removed by extending our algorithm to include a new adaptive P^3M technique, which we then introduce and present as a new `l1ap3m-hc` code. We optimize free parameters of adaptive P^3M to minimize force errors and the timing required to compute short range forces. We apply our codes to simulate small scale structure of the universe at redshift $z > 50$. We observe and analyze the formation of caustics in the structure and compare it with the predictions of semi-analytic models of structure formation. The current limits on neutralino detection experiments assume a Maxwell-Boltzmann velocity distribution and smooth spatial distribution of dark matter. It is shown in this thesis that inhomogeneous distribution of dark matter on small scales significantly changes the predicted event rates in direct detection dark matter experiments. The effect of spatial inhomogeneity weakens the upper limits on neutralino cross section produced in the Cryogenic Dark Matter Search Experiment.

Thesis Supervisor: Edmund Bertschinger
Title: Professor

Acknowledgments

I have greatly enjoyed working on this Thesis project. There are many people whom I would like to thank. First of all, I would like to thank my advisor Edmund Bertschinger for initiating this work, wise guidance on this project and for unconditional support and infinite patience. Without his ideas the results of this work would not be possible. I am grateful to Scott Burles and Alan Guth for their interest and for stimulating important parts of this project; to Scott Hughes for allowing me to use the MIT Astrophysics Beowulf cluster of computers for a long time above preference of much of his own work; to Pierre Sikivie, Max Tegmark and Paul Shapiro for stimulating discussions; and to Paul Schechter for being very helpful. Thanks to Bilge Demirkoz for stimulating discussions on the AMS experiment.

I must mention the faculty in my previous schools in Russia. Their advice, interest and personal involvement during the very early state of my interest in Astrophysics were invaluable to me.

While at MIT I have been very fortunate to be in frequent contact with (in truly random order) Aregjan Danogoulian, Maria Springer, Serhii Zhak, Marina Bevzushenko, Anatoly Dementyev, Konstantin Anikeev, Alexander Rakitine, Jeremy Schnittman, Valeriy Ivanov, Donglai Gong, Dmitry Pushin, Adam Bolton, Sergei Bashinsky and Jamie Portsmouth and many others. Thanks to Slobodan Jurac who was ready to help let the steam out for a break. Thanks to Judd Bowmann and Jake Hartman for their active involvement in the social activities which made big difference, and to the friendly pool of recent new graduate students who brought more life into the department.

Finally and foremost, I would like to thank my family for their constant care and support: my mother (Lidia), my brother Dima; my wife Olga with little daughter Veronica; and to my grandmother (Nadezhda) who was extremely happy when I came to MIT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Dark matter dominates the dynamics of our universe and constitutes most of the total matter content of the universe. Its mass is roughly six times larger than the mass in the baryonic form, which is observable by means of optical, X-ray, radio, and other telescopes.

First indications of the existence of dark matter were observations of the peculiar velocities in the nearby Virgo cluster by Zwicky in 1933, showing that the mass content of the cluster is much higher than that implied from the direct count of galaxies, their luminosities, and the assumption that the content of galaxies is dominated by star-like objects, whose approximate mass to light ratio is established. The lack of direct detection of dark matter implies that it does not participate in any of the non-gravitational interactions in which ordinary matter participates, or if it does, the interaction cross section is extremely low.

One can gain some insight on the properties of dark matter by looking at the dynamical evolution of stars in galaxies, whose dynamics are governed entirely by gravitation. Gravitational interaction participates in the formation of coherent long ranged structures such as spiral arms. In addition, we observe gravitationally bound globular and loose star clusters and finally, stars themselves. Even without being able to observe dark matter directly, we can deduce that its spatial distribution is very different from that for baryonic matter. Apparently, dark matter does not form star-like objects or replicate exactly the distribution of visible matter in galaxies, whose structure has formed as the result of collisional dissipation due to the electromagnetic interaction of baryonic particles. Dark matter has no pressure and it is collisionless, as far as the strong and electroweak interactions are concerned. The current spatial distribution of matter depends on the interaction in which it participates. The distribution of dark matter is entirely different from that of stars and other baryonic matter because its interactions are different.

Based on the hypothesis that dark matter consists of collisionless particles, one can deduce information on their spatial and velocity distributions. It follows from Zwicky's observations that the dark matter is overdense within the bounds of the Virgo cluster. Several decades after Zwicky's observations, the ideas about dark matter gained momentum when the measurements of rotation curves of gas around the center of galaxies had shown that the mass of galaxies themselves are dominated

by dark matter, meaning at least that the dark matter distribution has substructure on the scales of galaxies.

Since the discovery of dark matter, it has been very difficult to identify its nature. Early baryonic models suggesting that dark matter may be due to brown dwarfs or mini-black holes densely placed within galaxies have been ruled out by strong observational arguments. Dark matter remains undetected in the lab so far, but the idea that it consists of a new fundamental particle has received strong motivation, since weakly interacting particles are natural constituents in particle physics. There are currently two well-motivated particle physics candidates for dark matter: axions and neutralinos. Many conclusions of this Thesis are valid for both types of particles, however we will focus on neutralinos for simplicity of discussion, especially for the discussion of the experiments.

The study of the formation of cosmic structure in dark matter can be approached both analytically and computationally. The effect of baryonic matter on dark matter structure is limited since baryonic matter is a small fraction of the total matter and the dark matter interacts exclusively through gravitation. However, the effect of baryons becomes important at the late epochs of structure formation, when the baryonic matter has collapsed to form galaxies within the potential wells initially formed by dark matter.

Even at early stages of structure formation, the evolution of dark matter becomes extremely complicated. There are two phases in structure formation of collisionless matter, which can be described by the overdensity $\delta \equiv (\rho - \rho_0)/\rho_0$, where ρ is the local density and ρ_0 is the background (spatially averaged) density of the universe. Originally, the inhomogeneity (non-zero δ as a function of spatial coordinates) is thought to be produced as an outcome of random quantum-mechanical processes. Those fluctuations are stretched by inflation to scales much larger than the Hubble distance (the observable horizon). After inflation, the Hubble distance grows again until perturbations become smaller than the Hubble distance. After the universe becomes matter dominated at redshift $z = 4000$, the fluctuations grow in amplitude with $\delta \propto a$, while $\delta^2 \ll 1$. The linear theory of structure formation is based on applying the approximation $\delta^2 \ll 1$ to the dark matter fluid equations. Under gravity, the overdense regions accumulate more matter by accretion from underdense regions, whose average density decreases.

Overdense regions have a smaller dynamical time than the universe on average. Their evolution proceeds faster and if the average value δ of a perturbation is large enough, soon enough it will gravitationally decouple from the surrounding matter under the force of its own gravity and collapse as an independent object. The first dark matter structures in the Universe collapsed as early as a redshift $z \approx 350$, while the first galaxies are thought to have formed by $z \approx 10$. An analytical extension of linear solutions for density perturbations was developed by Zel'dovich in 1970. This extension provides approximate solutions for particle positions and density fields up to the time of the collapse of the earliest structure. Although there exist semi-analytic treatments of matter formation such as [32], [48], the structure of dark matter at the present time or even at time $z < 20$ can not be understood analytically. The Zel'dovich approximation breaks down after the time of the first collapse, and the

semi-analytic treatment can not describe the full range of phenomena of structure formation, such as hierarchical merging.

N-body simulations, which follow particle orbits using a programmable computer, provide valuable descriptions of the structure of dark matter at the later and nonlinear stage of structure formation, at which point the analytical treatment is qualitative or is not possible. In N-body simulations, dark matter is artificially sampled with particles, their gravitational field is calculated, and their positions are advanced step by step. N-body simulations have many advantages since they require few assumptions about the matter distribution. The limitations of the analytical approach are not applicable in N-body simulations, which however have their own limitations.

The limitations of N-body simulations include: 1) cosmological periodic boundary conditions designed to make a plausible model for the infinite universe, 2) discrete sampling of the mass of particles and their density perturbation spectrum, 3) absence of infinite dynamic range in any computed or sampled statistical quantity. However, with the increase of the number of particles that are treated by an N-body simulation, the influence of these limitations decreases, asymptotically approaching the precise implied model of the universe. That is why the problem of creating an efficient algorithm that enables simulations with a large number of particles is very important. The first part of this problem was solved in the 1960s and 1980s, when the Fast Fourier Transform technique and the particle-mesh method for computing gravity were introduced, respectively. The next two chapters of this thesis present an improved numerical N-body algorithm.

In the last twenty years other implementations of N-body codes were used in order to gain insight about structure formation. Since the dynamic range covered by simulations is finite, a simulation of a fixed volume typically does not allow dark matter mass resolution below a certain simulation-specific limit. For example, a typical simulation with a periodic simulation box of size 200 Mpc will not cover the mass resolution range on solar mass scales. The structures unresolved by the simulation particles may bring important additional physics into the dark matter. There have been many attempts in the simulations to obtain a plausible realization of the dark matter distribution in the universe purely from simulations, integrating all the way from analytically described random quantum initial fluctuations. Yet so far there have been few attempts in simulating very small structure on the universe, despite a number of theoretical predictions about the evolution of structure on these scales, first predicted by [5], [25] and developed in [48] and a series of works by P. Sikivie et al [47], [34] and [36]. The final chapter of this thesis will provide a plausible simulation of dark matter on small scales, test the semi-analytical models of [32] and [25], and subsequently investigate the effects of dark matter structures on the direct detection experiments searching for dark matter particles in laboratories.

Chapter 2

Scalable and Load Balanced P^3M Cosmological N-body Code

2.1 Introduction

Cosmological N-body simulations are the main tool used to study the dynamics of collisionless dark matter and its role in the formation of cosmic structure. They first became widely used 20 years ago after it was realized that the gravitational potentials of galaxies are dominated by dark matter. At the same time, theories of the early universe were developed for dark matter fluctuations so that galaxy formation became an initial value problem.

Although many of the most pressing issues of galaxy formation require simulation of gas dynamics as well as gravity, there is still an important role for gravitational N-body simulations in cosmology. Dark matter halos host galaxies and therefore gravitational N-body simulations provide the framework upon which one adds gas dynamics and other physics. Moreover, many questions of structure formation can be addressed with N-body simulations as a good first approximation: the shapes and radial mass profiles of dark matter halos, the rate of merging and its role in halo formation, the effect of dark matter caustics on ultra-small scale structure, etc.

In a cosmological N-body simulation, the matter is discretized into particles that feel only the force of gravity. A subvolume of the universe is sampled in a rectangular (not necessarily cubic) volume with periodic boundary conditions. In principle, one simply uses Newton's laws to evolve the particles from their initial state of near-perfect Hubble expansion. Gravity takes care of the rest.

In practice, cosmological N-body simulation is difficult because of the vast dynamic range required to adequately model the physics. Gravity knows no scales and the cosmological initial fluctuations have power on all scales. After numerical accuracy and speed, dynamic range is the primary goal of the computational cosmologist. One would like to simulate as many particles as possible (at least 10^{10} to sample galaxies well within a supercluster-sized volume), with as great spatial resolution as possible (at least 10^4 per dimension), for as long as possible (10^3 to 10^4 timesteps to follow the formation and evolution of structure up to the present day).

A single computer is insufficient to achieve the maximum possible dynamic range. One should use many computers cooperating to solve the problem using the technique of parallelization. In a parallel N-body simulation, the computation and memory are distributed among multiple *processes* running on different *nodes* (computers).[1] Unfortunately, ordinary compilers cannot effectively parallelize a cosmological N-body simulation code. A programmer must write special code instructing the computers how to divide up the work and specifying the communication between processes.

A parallel code is considered successful if it produces load-balanced and scalable simulations. A simulation is *load balanced* when the distribution of the effective workloads among the nodes is uniform. *Scalability* for a given problem means that the wall clock time spent by the computer cluster doing simulations scales inversely with the number of nodes used. Ideally, of course, the code should also be *efficient*: as much as possible, the wall clock time should be entirely devoted to computation.

At present, there are two main algorithms used for cosmological N-body codes: Tree and P^3M (see Bertschinger 1998 for review). The current parallel Tree code implementations include TreeSPH [14]), HOT [44], Gadget [49], and Gasoline [51]. Tree codes have the advantage of relatively easy parallelization and computing costs that scale as $N \log N$ where N is the number of particles. However, they have relatively large memory requirements.

The P^3M (particle-particle/particle-mesh) method was introduced to cosmology by [20] and is described in detail in [21] (see also Bertschinger & Gelb 1991). For moderate clustering strengths, P^3M is faster than the Tree code but it becomes slower when clustering is strong. This is because P^3M is a hybrid approach that splits the gravitational force field of each particle into a long-range part computed quickly on a mesh plus a short-range contribution computed by direct summation over close pairs. When clustering is weak, the computation time scales as $N_{\rm gr} \log N_{\rm gr}$ where $N_{\rm gr}$ is the number of grid (mesh) points, while when clustering is strong the computation time increases in proportion to N^2. The scaling can be restored to $N \log N$ using adaptive methods [13].

Currently there exist several parallel implementations of the P^3M algorithm, including the version of [24] for the (now defunct) Connection Machine CM-5 and the Hydra code of [37]. The Hydra code uses shared memory communications for the Cray T3E. There is a need for a message-passing based version of P^3M (and its adaptive extension) to run on beowulf clusters. This need motivates the present work.

The difficulty of parallelizing adaptive P^3M has led a number of groups to use other techniques to add short-range forces to the particle-mesh (PM) algorithm. The Tree and PM algorithms have been combined by [10] and [18] while [38] use a two-level adaptive mesh refinement of the PM force calculation. The FLASH code [26] has been extended to incorporate PM forces with multi-level adaptive mesh refinement.

When the matter distribution becomes strongly clustered, parallel codes based on PM and P^3M face severe challenges to remain load-balanced.

In general, P^3M and PM-based parallel codes suffer complications when the matter

[1] Because of the increasing availability of beowulf clusters, we consider only distributed memory parallelism.

becomes very clustered as happens at the late stages of structure formation. Most of the existing codes use a static one-dimensional slab domain decomposition, which is to say that the simulation volume is divided into slices and each process works on the same slice throughout, even when the particle distribution becomes strongly inhomogeneous. The GOTPM code uses dynamic domain decomposition, with the slices changing in thickness as the simulation proceeds, resulting in superior load balancing. However, even this code will break down at very strong clustering because it also uses a one-dimensional slab domain decomposition. The FLASH code uses a more sophisticated domain decomposition similar in some respects to the method introduced in the current chapter.

The motivation of the current work is to produce a publicly available code that will load balance and scale effectively for all stages of clustering on any number of nodes in a beowulf cluster. This chapter introduces a new, scalable and load-balanced approach to the parallelization technique for the P^3M force calculation. We achieve this by using dynamic domain decomposition based on a space-filling Hilbert curve and by optimizing data storage and communication in ways that we describe.

This chapter is the first of two describing our parallelization of an adaptive P^3M algorithm. The current chapter describes the domain decomposition and other issues associated with parallel P^3M. The next chapter describe the adaptive refinement method used to speed up the short-range force calculation.

The outline of this chapter is as follows. The serial P^3M algorithm (based on Gelb & Bertschinger 1994 and Ferrell & Bertschinger 1994) that underlies our parallelization is summarized in §2.2. Section 2.3 discusses domain decomposition methods starting with the widely-implemented static one-dimensional slab decomposition method. We then introduce the space-filling Hilbert curve and describe its use to achieve a flexible three-dimensional decomposition. Section 2.4 presents our algorithm for dynamically changing the domain decomposition so as to achieve load balance. Section 2.5 presents our techniques for organizing the particle data so as to minimize efficiency in memory usage, cache memory access, and interprocessor communications. In §2.6 we describe the algorithms used to parallelize the PM and PP force calculations. Section 2.7 presents code tests emphasizing load balance and scalability. Conclusions are presented in §2.8. An appendix presents an overview of the code and frequently appearing symbols, and another appendix briefly describe the routines used to map the Hilbert curve onto a three-dimensional mesh and vice versa.

2.2 Serial P^3M C-code and force calculation

In this section we summarize our serial cosmological N-body C implementation `p3m` based on an earlier serial Fortran implementation of P^3M by one of the authors. We discuss in detail the code units and aspects of the force calculation that are necessary for understanding the parallelization issues covered in the later sections.

2.2.1 Long and Short Range Forces and the Pairwise Force Law

Given the pairwise force $\mathbf{F_0}(\mathbf{r}_{12})$ between two particles of masses m_1 and m_2 and separation $\mathbf{r}_{12} = \mathbf{x}_2 - \mathbf{x}_1$, we define the interparticle force law profile $\mathbf{\Theta}_0(\mathbf{r}) \equiv \mathbf{F_0}(\mathbf{r}_{12})/(Gm_1m_2)$. For a system of many particles, the gravitational acceleration of particle i is $\sum_{j\neq i} Gm_j\mathbf{\Theta}_0(\mathbf{r}_{ij})$.

The required interparticle force law profile depends on the shape of the simulation particles. For point particles one uses the inverse square force law profile $\mathbf{\Theta}_0(\mathbf{r}) = -\mathbf{r}/r^3$. The inverse square force law is not used for simulation of dark matter particles in order to avoid the formation of unphysical tight binaries, which happens as a result of two-body relaxation [9]. For cold dark matter simulations many authors use the [41] force law

$$\mathbf{\Theta}_{\rm PL}(\mathbf{r},\epsilon) \equiv -\frac{\mathbf{r}}{(r^2+\epsilon^2)^{3/2}} \ , \tag{2.1}$$

where ϵ is the Plummer softening length. We take the Plummer softening length to be constant in comoving coordinates. With Plummer softening the particles have effective size ϵ. In a $\rm P^3M$ code, ϵ is usually set to a fraction of the PM-mesh spacing.

In a $\rm P^3M$ code, the desired (e.g., Plummer) force law is approximated by the sum of a long-range (particle-mesh or PM) force evaluated using a grid and a short-range (particle-particle or PP) force evaluated by direct summation over close pairs. The PM force $\mathbf{\Theta}_{\rm PM}(\mathbf{x}_i,\mathbf{x}_j)$ varies slightly depending on the locations of the particles relative to the grid (see Appendix A of Ferrell & Bertschinger 1994). The average PM force law $\langle\mathbf{\Theta}_{\rm PM}\rangle(\mathbf{r}_{ij})$ can be tabulated by a set of Monte-carlo PM-force simulations each having one massive particle surrounded by randomly placed test particles [6]. In practice, the mean PM force differs from the inverse square law by less than 1% for pair separations greater than a few PM grid spacings. For smaller separations, a correction (the PP force) must be applied. The total force is given by

$$\mathbf{\Theta}_{\rm P3M}(\mathbf{r}_{ij}) \equiv \mathbf{\Theta}_{\rm PP}(\mathbf{r}_{ij}) + \mathbf{\Theta}_{\rm PM}(\mathbf{x}_i,\mathbf{x}_j) \ . \tag{2.2}$$

Strictly speaking, the $\rm P^3M$ force is not translationally invariant and therefore depends on the positions of both particles. The $\rm P^3M$ force differs from the exact desired interparticle force profile $\mathbf{\Theta_0}$ by $\mathbf{\Theta}_{\rm Error}(\mathbf{x}_i,\mathbf{x}_j) \equiv \mathbf{\Theta}_{\rm P3M}(\mathbf{x}_i,\mathbf{x}_j) - \mathbf{\Theta_0}(\mathbf{r}_{ij}) = \mathbf{\Theta}_{\rm PM}(\mathbf{x}_i,\mathbf{x}_j) - \langle\mathbf{\Theta}_{\rm PM}\rangle(\mathbf{r}_{ij})$.

At large separations, both the PM-force and the required force reduce to the inverse square law (modified on the scale of the simulation volume by periodic boundary conditions). The PP-force can therefore be set to zero at $r \geq R_{\rm max}$ for some $R_{\rm max}$. The PP-correction is applied only for separations $r < R_{\rm max}$. The PM-force on the other hand is mainly contributed by remote particles.

2.2.2 Dynamic Equations and Code Units

The equation of motion of particles in a Robertson-Walker Universe is

$$\frac{d^2\mathbf{x}}{d\tau^2} + \frac{1}{a}\frac{da}{d\tau}\frac{d\mathbf{x}}{d\tau} = -\nabla_{\mathbf{x}}\phi \ , \tag{2.3}$$

where $\mathbf{x} \equiv \{x^0, x^1, x^2\}$ is the comoving position and τ is comoving (conformal) time. The potential ϕ satisfies the Poisson equation

$$\nabla^2_{\mathbf{x}}\phi = 4\pi G a^2 \delta\rho\,(\mathbf{x}, \tau) \ , \tag{2.4}$$

where $\delta\rho$ is the excess of the proper density over the background uniform density.

The equations take a simpler and dimensionless form in a special set of units that we adopt. The coordinates, energy and time in our code are brought to this form. Let us denote by tildes variables expressed in code units. Then for the units of time, position, velocity and energy (or potential), we write $d\tilde{t} = H_0 dt/a^2 = H_0 d\tau/a$, $\tilde{x} = x/\Delta x$, $\tilde{v} = v(a/H_0\Delta x)$ and $\tilde{E} = E(a/H_0\Delta x)^2$ or $\tilde{\phi} = \phi(a/H_0\Delta x)^2$, where a is the expansion factor of the universe, v is the proper velocity, H_0 is the Hubble constant and Δx is the cell spacing of the PM density mesh in our code (see Sec. 2.2.5) expressed in comoving Mpc. In these units, the equation of motion (2.3) reduces to

$$\frac{d\tilde{\mathbf{x}}}{d\tilde{t}} = \tilde{\mathbf{v}} \ , \quad \frac{d\tilde{\mathbf{v}}}{d\tilde{t}} = \tilde{\mathbf{g}} \equiv \tilde{m}\tilde{\mathbf{\Theta}}(\mathbf{r}) = -\nabla_{\tilde{\mathbf{x}}}\tilde{\phi} \ . \tag{2.5}$$

We choose units of mass so that the Poisson equation takes the following form in dimensionless variables:

$$\nabla^2_{\tilde{\mathbf{x}}}\tilde{\phi} = \frac{3\Omega_m a}{2}\delta\tilde{\rho} \ , \quad \delta\tilde{\rho} \equiv \delta\rho/\bar{\rho}_m \ , \tag{2.6}$$

where $\bar{\rho}_m = 3\Omega_m H_0^2/(8\pi G)$ is the proper mean matter density. Particle masses are made dimensionless by $\tilde{m} = m/[\bar{\rho}_m(a\Delta x)^3]$. The dimensionless total mass of all the particles is $\tilde{M}_{\rm tot} = N_{\rm gr}$ where $N_{\rm gr}$ is the total number of PM mesh points. Periodic boundary conditions are assumed in each dimension so that a finite volume simulation represents a small portion of a universe that is homogeneous on larger scales.

As a check on overall code accuracy, we monitor global energy conservation by integrating the Layzer-Irvine equation, which in code units takes the form

$$\frac{d}{d\tilde{t}}\left(\tilde{E}_k + \tilde{E}_g\right) = \frac{\tilde{E}_g}{a}\frac{da}{d\tilde{t}} \ , \tag{2.7}$$

where

$$\tilde{E}_g \equiv -\frac{1}{2N_{\rm gr}}\sum_{i,j}\tilde{m}_i\tilde{m}_j\int_{r_{ij}}^{\infty}\tilde{\mathbf{\Theta}}(r)\,dr \quad \text{and} \quad \tilde{E}_k \equiv \frac{1}{N_{\rm gr}}\sum_i \frac{\tilde{m}_i\tilde{v}_i^2}{2} \tag{2.8}$$

are the dimensionless gravitational and kinetic energies in the simulation. Note that in a Robertson-Walker background, the Hamiltonian is time-dependent and so the energy is not conserved [8]. However, we can integrate equation (2.8) to get a quantity

that should remain constant as the simulation progresses,

$$\tilde{E}_{\rm con} \equiv \int d\tilde{E}_k + \int d\tilde{E}_g - \int \tilde{E}_g \, d\ln a \, . \tag{2.9}$$

2.2.3 Particle Data Structure and Layout

A particle is represented in both our serial and parallel codes by a structure, defined as

$$\begin{aligned} \texttt{typedef struct part\_t } \{ & \texttt{float x0, x1, x2, mass, g0, g1, g2;} \\ & \texttt{int id; float v0, v1, v2; } \} \texttt{ part\_t} \, . \end{aligned} \tag{2.10}$$

The size of the structure is 44 bytes on 32 bit machines. The structure contains three positions, the mass, accelerations and velocities of the particles along the three spatial Cartesian directions, all made dimensionless by the choice of units described above. In addition, the integer `id` is used to tag particles. This number can be arbitrary and is not used anywhere in force calculations or particle propagation. In the serial code, the particles are stored in memory simply as an array with base pointer `pa` and end pointer `pa_f` $=$ `pa` $+ N$, where N is the total number of particles in the simulation volume. To scan all the particles, e.g. for their imaging, we loop over all the pointers `p` within the range `p` $\in$ [`pa`, `pa_f`). The particle masses are not required to be equal to each other in general.

2.2.4 Particle Integration and Timestep Criterion

All the particles in the code are positioned within the simulation box of size $L^i = n^i \Delta x$, where $i \in \{0, 1, 2\}$ labels the spatial dimension. (We allow for unequal lengths with n^i/n^j equalling a ratio of small integers.) Periodic boundary conditions are applied to bring particles that move outside back into the simulation volume. We currently use a Drift-Kick-Drift (DKD) leapfrog integrator scheme [43, 45] to integrate the equations of motion (2.5) for the particles:

$$\begin{aligned} & \mathbf{x}_{n+1/2} = \mathbf{x}_n + \tfrac{1}{2}\mathbf{v}_n \, \Delta t \\ & \text{Force calculation } \mathbf{g}_{n+1/2} \\ & \mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{g}_{n+1/2} \, \Delta t \\ & \mathbf{x}_{n+1} = \mathbf{x}_{n+1/2} + \tfrac{1}{2}\mathbf{v}_{n+1} \, \Delta t \, , \end{aligned} \tag{2.11}$$

where the subscripts denote timesteps. [31] discuss the accuracy and stability of this scheme. Note that the P^{3}M force calculation is needed only once each timestep.

All integrators have advantages and limitations. For our problem, which can be expressed as a continuum Hamiltonian time evolution, the leapfrog integrator of equation (2.11) is a good choice, since with a constant timestep it is *symplectic*. A symplectic integrator preserves the Poincaré integral invariants and follows the time evolution under a discrete Hamiltonian that is close to the continuum Hamiltonian

of interest. The difference between the discrete and continuum Hamiltonian or the discrete integrator error is itself a Hamiltonian. When the error is a Hamiltonian, and is sufficiently small, according to the KAM theorem [3] the difference between the Hamiltonian paths evolved by the two Hamiltonians is a set of finite measure. Therefore most of the structure of the Hamiltonian flow evolved by the continuum Hamiltonian will be preserved when evolved by the discrete Hamiltonian with the symplectic integrator. Most of the stable orbits in the continuum Hamiltonian system will remain stable under the discrete Hamiltonian evolution and vice versa.

Higher order symplectic integrators for Hamiltonian evolution can be constructed using the method of [52], which requires more force evaluations per timestep. In general, a N-th order symplectic integrator requires at least $N-1$ force evaluations per complete timestep.

In a cosmological simulation, particles become more clustered with time. It is not practical therefore to have a fixed value of the timestep for the whole simulation. Currently we advance equation (2.11) with the same value of timestep Δt for all the particles but allow it to change with time. The choice for Δt is based on the current particle data and therefore depends on the phase space variables. Consequently, equation (2.11) is no longer an exact symplectic map. Nevertheless, it remains in practice well-behaved provided the timestep varies sufficiently slowly.

The timestep must at least satisfy the leapfrog stability criterion $(da/dx)_{\max}(\Delta t)^2 < 4$ given by equation (4-42) of [31]. This stability requirement is essentially equivalent to the constraint that the global timestep must be small enough not to exceed the local dynamical time at any point within the simulation box, $\Delta t \sim \sqrt{\eta_t/(G\rho_{\max})}$ where η_t is a dimensionless constant. The density is somewhat expensive to obtain, but given the particle accelerations and using the approximation $g \sim GM/r^2 \sim G\rho r^3/r^2 \sim G\rho\epsilon$, we have now, expressed in code units,

$$\Delta\tilde{t} = \sqrt{\frac{\eta_t\,\tilde{\epsilon}}{\tilde{g}_{\max}}}\,, \tag{2.12}$$

where $\tilde{\epsilon}$ is the dimensionless Plummer softening length (see § 2.2.1), η_t is a free parameter in the code usually set to a small value such as $\eta_t = 0.05$, and $\tilde{g}_{\max}$ is the maximum acceleration of a particle in the simulation box in code units. This criterion is conservative in assuring that the orbits of all particles are well sampled.

To further improve the integration technique, one may consider adaptive integrators, with individual particle timesteps changing according to the local dynamical time at the given position within the simulation volume [42]. On the other hand, one may consider higher order symplectic integrators, which would require more force evaluations per timestep. Some of the non-symplectic higher order integrators, such as Runge-Kutta, are known not to preserve the Hamiltonian flow structure even with fixed timesteps. For example, it can be shown by integration of Kepler orbit that the popular fourth order Runge-Kutta integrator yields a divergent orbit very quickly. On the other hand, for second order integrators, the DKD scheme shows stable orbits with errors behaving as small perturbations as expected on the basis of KAM theo-

rem. Figure 1 of [42] shows the phase space portrait of a Hamiltonian system evolved by Runge-Kutta vs. DKD, in which the trajectory of the former deviated from the analytical solution much faster than the trajectory of the second order leapfrog.

In this chapter we adopt the leapfrog integrator with variable timestep (set to the same value for all particles), leaving the implementation of a higher order symplectic integrator and individual particle timesteps for future work.

2.2.5 Particle-Mesh Force Calculation

The particle–mesh (PM) force is the long-range force that can be computed using Fast Fourier Transforms (FFT). In our code, we use the FFTW Fourier transform implementation [22] and the PM algorithm of [23]. For large total number of particles N in the simulation box, the PM force computation is faster than the direct summation method, requiring only $\propto N_{\rm gr} \log N_{\rm gr}$ operations in total ($N_{\rm gr} \equiv n^0 n^1 n^2$ is the number of PM grid points), as opposed to $O(N^2)$.

The rectangular PM-density mesh is allocated for the whole simulation volume in the serial code. This grid is to be filled with the density values interpolated from the particles nearby. [31] discuss a number of methods for the density interpolation with increasing smoothness, ranging from Nearest Grid Point (NGP) to Cloud-in-Cell (CIC) to the Triangular-Shaped Cloud (TSC) method. The highest of accuracy of these is given by the TSC interpolation scheme and that is the scheme we have implemented. As shown by [31], an interpolation of the mass value from a particle at position $\tilde{\mathbf{x}}$ to a grid point at position $\tilde{\mathbf{x}}_{\rm gr}$ within the PM mesh and vice versa takes place if and only if

$$\max_{i=\{0,1,2\}} \left|\tilde{x}^i - \tilde{x}^i_{\rm gr}\right| \leq L_{\rm SCH} \ , \tag{2.13}$$

where the absolute value is taken with the proper account for the boundary conditions, and $L_{\rm SCH}$ is the window function domain locality length, specific to the interpolation scheme used, e.g. $L_{\rm NGP} = \frac{1}{2}$, $L_{\rm CIC} = 1$ and $L_{\rm TSC} = \frac{3}{2}$. In our code we have $L_{\rm SCH} = L_{\rm TSC}$.

There are several steps involved for one PM force calculation:

1. Density interpolation: Masses of particles are interpolated to a rectangular density mesh of grid points using a forward TSC interpolation scheme as illustrated by the left Figure 2-1. Details are given on pp. 142–146 of [31] and equation (A.16) of [23].

2. The mesh density is Fourier transformed to the complex domain.

3. The force is computed in the complex domain using a pretabulated Green's function given by equation (A.14) of [23].

4. The mesh force field is inversely Fourier transformed to return to the real domain.

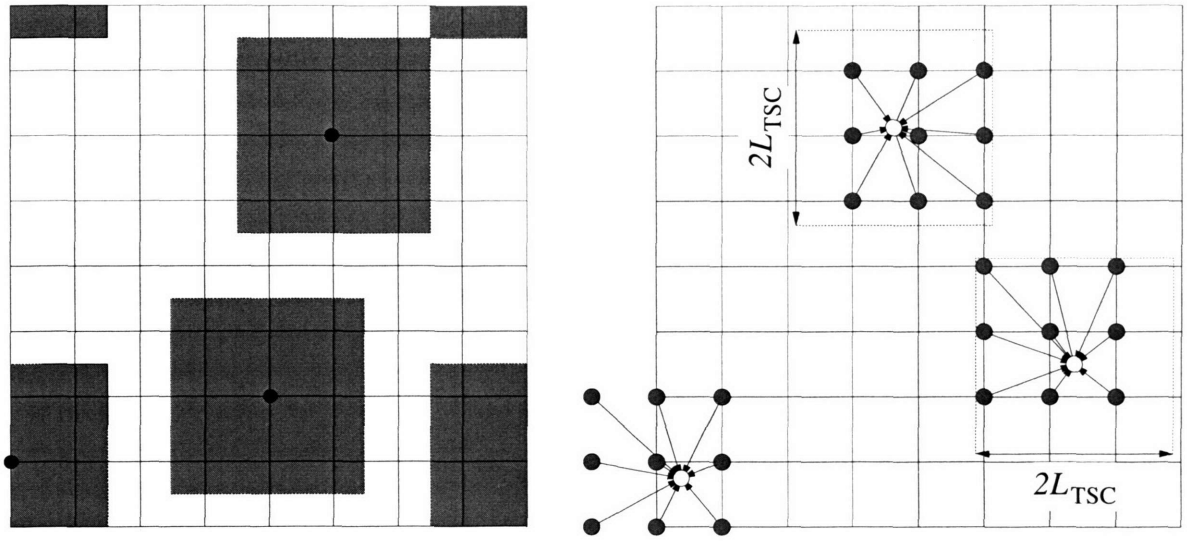

Figure 2-1: Left: Density interpolation from particles to grid points PM Step 1. To interpolate the density to a given grid point one needs to add contributions from all of the particles inside the shaded box of length $2L_{\rm TSC}$ centered on the grid point. Right: force interpolation from grid points to particles PM Step 5. To get the force on a given particle (open circle), force values are used from all of the surrounding grid points.

5. Force interpolation: Forces are interpolated from the force mesh to particles using a backward TSC interpolation scheme, as shown in the right Figure 2-1. This step is opposite to Step 1.

In Step 5, information flows in exactly the opposite direction as Step 1. Only the same grid points satisfying equation (2.13) that acquired their density values from the particles in Step 1 are used for the interpolation of the forces to only the same particles in Step 5. If an exchange of the information between a grid point and a particle ever occurs, *it has to be both ways.* This point will be very useful when we discuss density and force grid messages for the parallel code in §2.6.1.

The timing of the PM force evaluation scales as

$$t_{\rm PM} \propto A\,N + B\,N_{\rm gr}\log(N_{\rm gr})\ , \tag{2.14}$$

where the first term is due to the density and force interpolation and the second is due to the Fast Fourier Transform. The coefficients A and B do not depend on N and $N_{\rm gr}$. The coefficient A depends on the interpolation scheme used. For the TSC interpolation scheme in $d = 3$ dimensions, the density is always interpolated from a particle to the $(2L_{\rm TSC})^d = 27$ nearby grid points satisfying the condition (2.13). During the force interpolation, the inverse occurs three times: once for each of the three spatial dimensions. The factor of 4×27 therefore enters into an expression for A when TSC interpolation is used. The coefficient B is independent of A and is given by the existing benchmarks for the FFTW implementation [22].

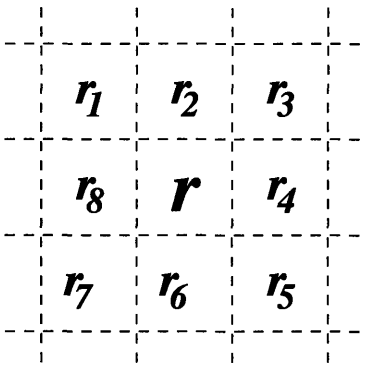

Figure 2-2: The cells $j_0 \ldots j_3$ constitute the 4 cells needed for PP force calculation in two dimensions, $B^{\rm PP} = \{j_0, j_1, j_2, j_3\}$.

2.2.6 PP Force Calculation and the Chaining Mesh

In order to calculate the short range force, we must first find all the pairs separated by less than $\tilde{R}_{\rm max}$. This is accomplished using a fast linked-list sorting procedure [31]. At the start of a simulation the whole simulation volume is partitioned into rectangular *chaining mesh* cells whose spacings in dimension i are constrained by

$$\Delta\tilde{x}_c^i \geq \tilde{R}_{\rm max} \ . \tag{2.15}$$

Given this constraint, for any particle in any chaining mesh cell, only the particles within the same or one of the adjacent chaining mesh cells need be included in the short range force calculation, since the PP force is zero for separations greater than $\tilde{R}_{\rm max}$. Choosing the smallest possible value satisfying equation (2.15), this leads to

$$\Delta\tilde{x}_c^i = \frac{\tilde{L}^i}{n_c^i} \ , \quad \text{where} \quad n_c^i = \left[\tilde{L}^i / \tilde{R}_{\rm max} \right] \ , \tag{2.16}$$

where the square brackets signify taking the integer part. At the start of the run, we sort all the particles into chaining mesh cells occupying the 3D volume and form linked lists of particles belonging to each cell. Each chaining mesh cell then contains the root of the linked list to all the particles within that cell.

In order to apply a short range force correction to a particle p within the simulation volume, we access particles contained within the same cell as well as the particles within the $3^d - 1 = 26$ surrounding chaining mesh cells. Since the short range correction procedure is applied for each pair of particles within the simulation volume, we need to traverse only half of the surrounding cells, as illustrated in Figure 2-2. For a given chaining mesh cell j, let $N_{\rm CM}(j)$ be the number of particles within the cell and $B^{\rm PP}(j)$ be the set of the $(3^d - 1)/2$ surrounding cells used for the short range force calculation. The number of floating point operations needed in order to apply the short range force correction for every particle within the simulation volume scales

Table 2.1: Dominant memory requirements of the serial p3m code

	Memory size	Memory size, for a 32^3 P^3M simulation, in bytes.
Particle array	4 bytes $\times$ $11N$	1,441,792
Particle linked list	4 bytes $\times$ $2N$	262,144
Chaining mesh	4 bytes $\times$ $n_c^0 n_c^1 n_c^2$	5,324
Green's function	4 bytes $\times$ $n^0 n^1 (n^2/2+1)$	69,632
Density and Force meshes	4 bytes $\times$ $2 \times n^0 n^1 (n^2+2)$	278,528
Total	4 bytes $\times$ $(13N + 5n^0 n^1 (n^2/2+1) + n_c^0 n_c^1 n_c^2)$	2,057,420

as

$$t_{\rm PP} \propto \sum_j N_{\rm CM}(j) \left[\frac{1}{2} N_{\rm CM}(j) + \sum_{j' \in B^{\rm PP}} N_{\rm CM}(j') \right] . \tag{2.17}$$

The PP force calculation takes a lot of time when particles are highly clustered because of the quadratic dependence on numbers.

2.2.7 Memory requirements

The total memory requirement for the serial code consists of several significant parts listed in Table 2.1, where the variables n^i, n_c^i and N are defined in Table 2.5 of Appendix 2.9. Using the serial N-body code with an average of p particles per PM gridpoint, the total memory requirement for a P^3M code is

$$M_{\rm tot} = \left(13p + \frac{5}{2} + \frac{1}{\tilde{R}_{\rm max}^3} \right) n^0 n^1 n^2 \times 4 \text{ bytes}$$

The maximum amount of memory available for dynamic allocation for a 32-bit machine in Unix is 2 GB. In practice the amount of memory available for our application is about 30% smaller. For a simulation having one particle per density mesh cell with a cubic grid, $M_{\rm tot} = (31/2)\, N_{\rm gr}^3 \times 4$ bytes. The maximum problem size for such a simulation with the upper limit on total memory of 1.4Gb is $N_{\rm gr} = 296^3$. This severe limitation on problem size is avoided using the parallel code described in the rest of this chapter.

2.3 Hilbert Curve Domain Decomposition

In order to perform simulations with more than 296^3 particles and gridpoints, we distribute the computation to multiple processors of a parallel computer. We are using the Single Program, Multiple Data (SPMD) model in which one program runs

on multiple processors which perform computations on different subsets of the data. The first decision to be made is how to distribute the data and computation. The computational volume is divided into parts called domains and the memory and computation associated with each domain is assigned to a different parallel process.

The problem of domain decomposition is to decide how to partition the computational volume into domains. As we will see, there are a number of considerations that enter this decision. This section first describes the simplest method, one-dimensional static domain decomposition, which is well suited for spatially homogeneous problems but not for strongly clustered N-body simulations. We then introduce the Hilbert curve method of dynamic domain decomposition used in our parallel code.

We use the word *process* to refer to one of the instances of our parallel program being applied to the data in its domain. A process may correspond to one CPU (or one virtual CPU, in the case of hyperthreading) or there may be multiple processes on one CPU.

2.3.1 Static Slab Domain Decomposition

In a static slab domain decomposition, the volume is divided by fixed planes with equal spacing. This it the method used, for example, in the FFTW Fast Fourier Transform [22]. It is well suited for problems in which the computation is uniformly distributed over volume. A variation on this method is to use a two-dimensional lattice of columns instead of a one-dimensional lattice of slabs.

Several groups have implemented static domain decomposition in parallel N-body codes based on PM or P^3M (see §2.1). As a first step, we developed our own implementation `llpm-sl` of the static slab domain decomposition Particle-Mesh N-body code.

A static slab or any other static particle domain decomposition is a good strategy when the number density distribution of particles across the simulation box is nearly uniform and each slab contains approximately the same number of particles to process each timestep. However, gravitational instability destroys the spatial uniformity leading to serious inefficiency. As particle clusters grow, the memory and computational resources of the processes containing the largest clusters (e.g. processes 1, 2, 3, 8, and 9 in Figure 2-3) grow quickly. Other processes finish their work and have to wait idle. Worse, the heavily loaded processes may run out of memory causing paging to disk. The inevitable result is that the computation becomes unbalanced and the code grinds to a halt (see the timing results in §2.7 for a 512^3 test run). The same problem will arise in any gravitational N-body code that uses static domain decomposition.

Such a situation, when the performance of the cluster degrades as a result of hugely varying workloads, is called work *load imbalance*. In the remainder of this section we introduce an alternative method of dynamic domain decomposition that solves the load imbalance problem for strongly clustered systems.

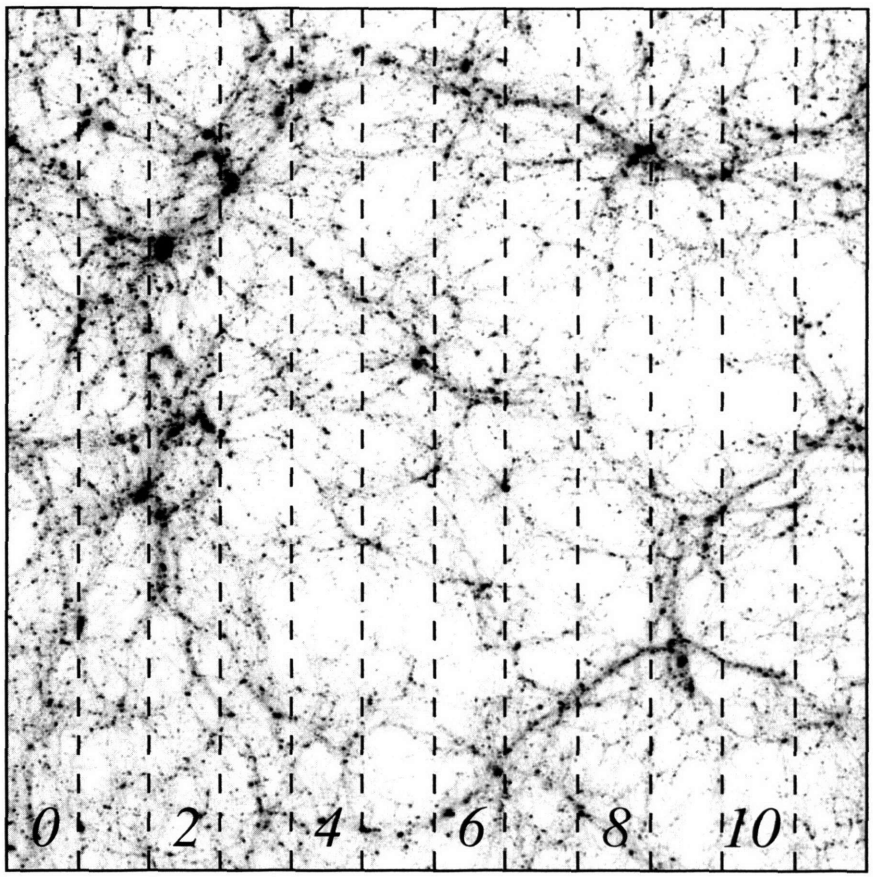

Figure 2-3: Sample particle distribution with 12 computing processes and a static slab domain decomposition. Processes 1, 2, 3, 8, and 9 have significantly more particles than the others.

2.3.2 Dynamic Domain Decomposition with a Hilbert Curve

As we have seen from the slab domain decomposition example in the previous section, it is important for an N-body code to load balance. We solve the load balancing problem by the implementation of dynamic particle domain decomposition defined by a Hilbert space-filling curve as suggested by [40]. Domain decomposition methods based on Morton ordering (a different space-filling curve) have been used by [44] and [26].

The Hilbert curve (HC) is a fractal invented by the German mathematician in 1891 and is one of the possible space-filling curves that completely fill a cubic rectangular volume. A unique HC is defined for any positive integer m (the *HC order*), and dimensionality d, for which the HC will fill each cell of a d-dimensional cube of length 2^m. For $d = 2$, examples are given in Figures 2-4 (with $m = 4$) and 2-5 (with $m = 3$). The HC provides a bijective (one-to-one) mapping between the index h along the curve (the *HC index*) and the cell within the volume. In our code the mapping was provided by the Hilbert curve implementation of [39] (see Appendix 2.10). The real

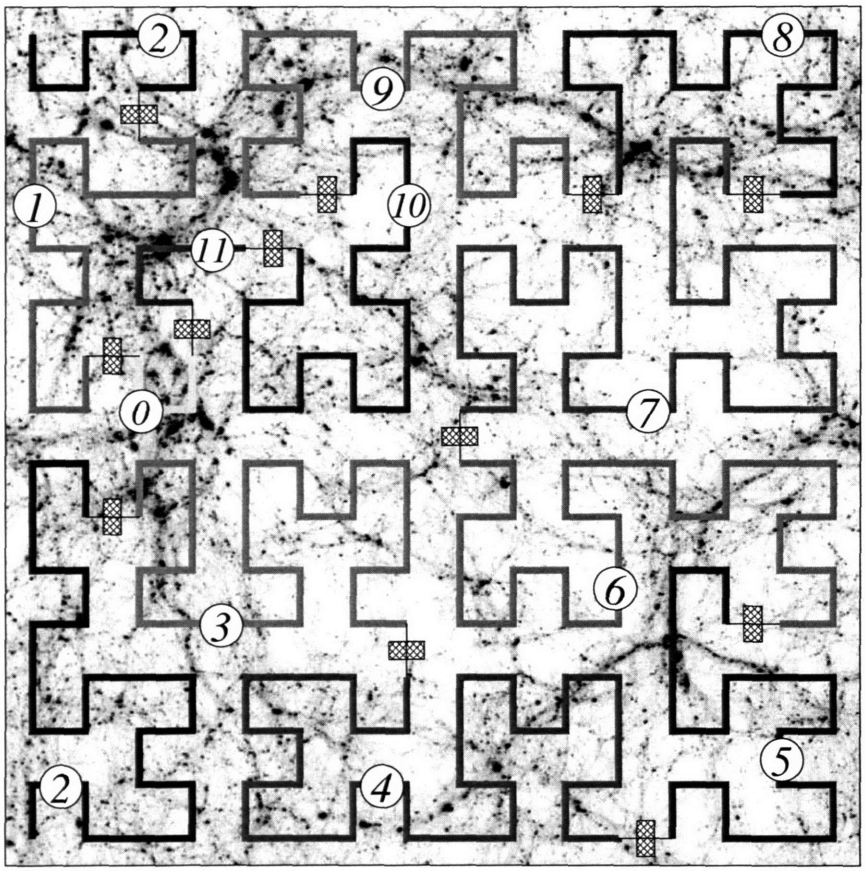

Figure 2-4: The same matter distribution as in Fig. 2-3, but now superimposed by a Hilbert curve. The Hilbert curve is divided into 12 colored segments separated by cross-hatched bars and labelled by the circled numbers. During the run the partitions will move along the Hilbert curve so that each process will have approximately the same amount of work to do. In a real simulation the Hilbert curve is divided into much finer segments.

simulation volume and the space filling curve we use are in fact three-dimensional but the two-dimensional case is used in the figures throughout the chapter in order to simplify the presentation.

The main idea of Hilbert curve domain decomposition is to take a three-dimensional volume with inhomogeneous workload and to convert it into a one-dimensional curve that is easily partitioned into approximately equal workloads. The key advantage compared with slab decomposition is that the Hilbert curve method breaks up the problem into 2^{md} chunks of work with $d = 3$ instead of $d = 1$. With much finer granularity it is possible to load balance extremely inhomogeneous problems. In addition, the Hilbert curve minimizes communication between processes, as we show below.

The Hilbert curve has the following properties:

1) *Compactness*: it tends to fill the space very compactly. A set of cells defined by a continuous section of a HC tends to be quasi-spherical, having small surface to

volume ratio. One can approximate the surface to volume ratio of any continuous segment of n cells along the three-dimensional Hilbert curve with

$$\text{S.V.R.}(n) \approx \frac{4.8}{n^{1/3}} , \tag{2.18}$$

which decreases with the increasing n. This approximation is crude at small n. The maximum possible ratio S.V.R.$(1) = 26$ is reached for $n = 1$, since one volume cell is surrounded by 26 adjacent surface cells.

2) *Locality*: the successive cells along the curve are mapped next to each other within the mesh;

3) *Self-similarity*: the curve is self-similar on different scales. It can therefore be extended to arbitrarily large size.

Figure 2-4 demonstrates the bijective mapping of 16×16 cells in a two-dimensional computational volume onto the indexed Hilbert space-filling curve. The curve visits each cell of the simulation volume exactly once. By connecting the two ends of the curve, the curve has the topology of a circle. By introducing n_{pr} partitions along the circle (the *partitioning state*) each being ascribed to one of the n_{pr} processes in the parallel code, we specify the particle domain decomposition of the whole simulation volume into n_{pr} *Local Regions*, each consisting of the cells along the curve between two adjacent partitions and being assigned to one of the n_{pr} processes. Let us denote the local region of process i defined by the partitioning state and the Hilbert curve by L_h^i.

As we see, the space-filling curve provides an easy way of bookkeeping for decomposition, since the local domains of each process are completely specified by the Hilbert curve setup and the n_{pr} numbers that specify the partitioning state.

The surface to volume ratio of local domains defined by the continuous segments of the Hilbert curve is small due to the compactness property of the Hilbert curve. This is the primary reason for choosing a Hilbert curve as the space filling curve for our domain decomposition. The small surface to volume ratio significantly speeds up the reassignment of particles crossing the L_h boundaries (§ 2.5.1) and the PP-force computation (§ 2.6.3). In the $N = 800^3$ run presented in §2.7.3, the surface to volume ratio was on average 0.1 for the domains of voids. In the Hydra code [37], using a static 7×7 two-dimensional cyclic domain decomposition, the surface to volume ratio is $(4 \times 7)/(7^2) \approx 57\%$, leading to more than five times as much communication cost for the particle advancement and the PP-calculation in comparison to our algorithm.

2.3.3 Hilbert Curve Initialization

At the beginning of a simulation we set up the Hilbert curve completely using the functions of [39] with an appropriate choice of the HC mesh parameters. Only one parameter, the Hilbert curve order m, is needed to completely specify the geometry of a Hilbert curve filling an entire d-dimensional cube of volume $(2^m)^d$, which we will call *the complete HC-mesh*. Adding more parameters — the *HC mesh cell spacings* dx_h^i , $i \in \{1, 2, 3\}$, the curve starting point in the simulation volume, and the curve

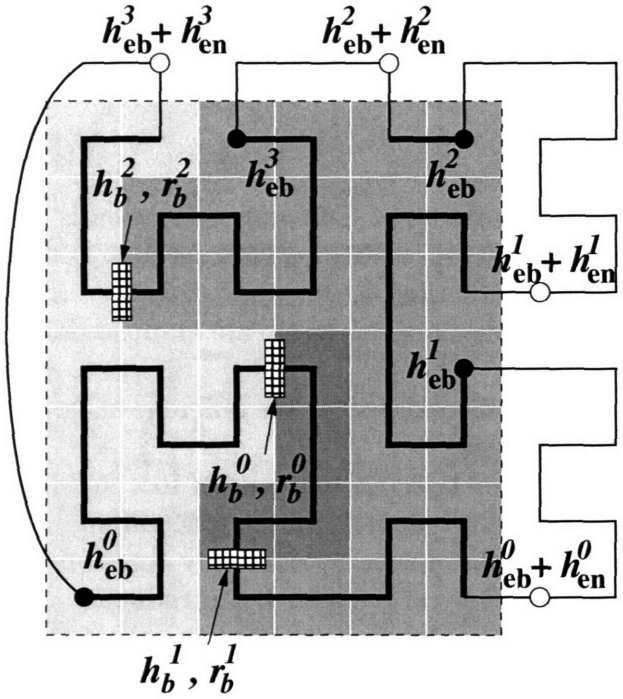

Figure 2-5: Structure of the domain decomposition. A two-dimensional Hilbert curve (*solid line*) of order $m = 3$ fills a $6dx_h^0 \times 7dx_h^1$ simulation box (*dashed line*). By connecting the ends of the Hilbert curve, the resulting curve has a circular topology. The number of processes is $n_{\rm pr} = 3$, so there are 3 partitions along the circle indicated by the cross-hatched bars. We have $h_{\rm b}^i = \{11, 14, 58\}$, $h_{\rm n}^i = \{3, 44, 17\}$ and $r_{\rm b}^i = \{11, 14, 38\}$, $r_{\rm n}^i = \{3, 24, 15\}$.

orientation — completely determines the Hilbert curve within the simulation volume.

While the real N-body simulation volume and the space filling curve are three dimensional, two dimensional examples are used in figures throughout this chapter solely to simplify the presentation.

We use the Hilbert curve in our code only to specify the domain decomposition for particle storage and computation. The domain decomposition does not affect any physical values computed. The choice of the Hilbert curve order m in our code is made based solely on the parallel code performance considerations. From the point of view of improving the resolution for particle domain decomposition, higher m is preferred. On the other hand each local region cell costs additional memory, favoring lower m. For a P^3M simulation, in order to simplify the force calculation, we choose the HC mesh cells to coincide with the PP chaining mesh cells:

$$\Delta\tilde{x}_h^i \equiv \Delta\tilde{x}_c^i \ , \quad n_h^i \equiv n_c^i \ . \tag{2.19}$$

While the complete HC-mesh is a cube of length 2^m cells, the chaining mesh length does not have to be a power of two. Therefore we choose the HC order m to be the

smallest integer satisfying

$$2^m \geq n_h^i \ , \quad i = \{x, y, z\} \ . \tag{2.20}$$

From equations (2.19) and (2.20), the complete chaining mesh is just a subset of the complete HC mesh. If $n_h^i = 2^m$ for all $i = 0 \ldots (d-1)$, as in Figure 2-4, the curve completely fits the simulation volume and the two coincide. If $2^m > n_h^i$ for some i, the complete Hilbert curve mesh covers an extra space outside the chaining mesh of the simulation volume as in Figure 2-5, containing the chaining mesh as a subset. We will refer to this submesh as the *Simulation Volume HC mesh* or simply as the *Hilbert curve mesh* where the context is clear.

Since the cells of the HC outside the simulation volume are irrelevant, they do not take memory and their HC indices are irrelevant too. Let us introduce a *raw HC index* along the curve. For a HC mesh cell $\mathbf{c}$ which belongs to the simulation volume, we define the *raw HC index* r, $r \in [0, n_h^0 n_h^1 n_h^2)$, as the number of HC cells that the curve spent within the simulation volume since its starting point (HC index $h = 0$). In other words, while the HC index is incremented each cell along the curve, the HC raw index is incremented only at the cells along the curve that belong to the simulation volume.

The mapping between the HC index h and the HC raw index r is specified completely by the *table of HC entries*. Each entry contains the HC index of an entry point $h_{\rm eb}^k$ of the curve into the simulation volume and the number of consecutive HC cells that the HC spends within the simulation volume $h_{\rm en}^k$ before the next exit. Let K be the number of entries in the HC table, and let $K \equiv 1$ if the HC mesh fits the simulation box exactly [$n_h^i = 2^m$ for each $i = 0 \ldots (d-1)$]. Because the Hilbert curve visits all the cells in the simulation box, we have

$$n_h^0 n_h^1 n_h^2 = \sum_{k=0}^{K-1} h_{\rm en}^k \ . \tag{2.21}$$

We denote the mapping of a cell $\mathbf{c}$ in the simulation box into its HC raw index r by $\mathcal{H}_{\rm r}(\mathbf{c})$.

Figure 2-5 gives an example of ${n_h}^0 \times {n_h}^1 = 6 \times 7$ simulation volume mapped by an 8×8 Hilbert curve ($m = 3$) in two dimensions ($d = 2$). Table 2.2 lists all the cells of the complete HC mesh $h = [\,0, 2^{dm} - 1)$ along with the raw index of those of them that belong to simulation volume HC mesh. The HC table of $K = 4$ entries is

$$\{\{h_{\rm eb}^k, h_{\rm en}^k\} : k = 0 \ldots (K-1)\} = \{\{0, 20\}\,\{28, 8\}, \{45, 2\}, \{50, 12\}\} \ . \tag{2.22}$$

The simulation volume contains 42 cells, in agreement with equations (2.21) and (2.22).

The space locality of the HC as a curve filling the simulation volume is lost if $K \neq 1$. Once the curve exits the simulation volume, the next entry back into the simulation volume may be far away (see Fig. 2-5). The resulting L_h may therefore consist of several disjoint parts, each having a surface to volume ratio given by equation (2.18).

Table 2.2: Example of mapping of the 2-dimensional simulation volume shown in Fig. 2-5 with a Hilbert curve. For each HC index h, the coordinates of the cell (c^0, c^1) are shown as well as the HC raw index if the cells belongs to the simulation volume.

h	c^0	c^1	r	h	c^0	c^1	r	h	c^0	c^1	r	h	c^0	c^1	r	h	c^0	c^1	r	h	c^0	c^1	r
0	0	0	0	11	3	2	11	22	7	1	–	33	4	5	25	44	5	7	–	55	2	5	35
1	1	0	1	12	3	1	12	23	6	1	–	34	5	5	26	45	5	6	28	56	1	5	36
2	1	1	2	13	2	1	13	24	6	2	–	35	5	4	27	46	4	6	29	57	1	4	37
3	0	1	3	14	2	0	14	25	7	2	–	36	6	4	–	47	4	7	–	58	0	4	38
4	0	2	4	15	3	0	15	26	7	3	–	37	7	4	–	48	3	7	–	59	0	5	39
5	0	3	5	16	4	0	16	27	6	3	–	38	7	5	–	49	2	7	–	60	0	6	40
6	1	3	6	17	4	1	17	28	5	3	20	39	6	5	–	50	2	6	30	61	1	6	41
7	1	2	7	18	5	1	18	29	5	2	21	40	6	6	–	51	3	6	31	62	1	7	–
8	2	2	8	19	5	0	19	30	4	2	22	41	7	6	–	52	3	5	32	63	0	7	–
9	2	3	9	20	6	0	–	31	4	3	23	42	7	7	–	53	3	4	33				
10	3	3	10	21	7	0	–	32	4	4	24	43	6	7	–	54	2	4	34				

Since the surface to volume ratio of a segment of HC decreases with increasing number of cells in the segment, taken together those subsegments have bigger surface to volume ratio than one big segment of the HC of same volume. A smaller value of surface-to-volume ratio reduces the communication cost of PP-force calculation by approximately the same factor (see § 2.6.3).

2.3.4 Local Regions and Partitioning State

To completely specify local regions L_h^i of each process $i \in [0, n_{\rm pr})$, we introduce $n_{\rm pr}$ partitions along the curve. A bottom partition of the process i is set by the raw HC index $r_{\rm b}(i)$ (also denoted $r_{\rm b}^i$) of the cell directly above the partition along the HC. In Figure 2-5, for example, the entire domain is divided between three worker processes by the three partitions with indices $r_{\rm b}^i = \{11, 14, 38\}$.

In general, a *partitioning state* and therefore all local regions L_h^i, $i \in [0, n_{\rm pr})$ are completely specified by a set of $n_{\rm pr}$ numbers $\{r_{\rm b}(0),\, r_{\rm n}(i)\colon\, i = [0, n_{\rm pr} - 1)\}$, where $r_{\rm n}^i$ is the spacing between the partitions i and $i+1$. This implies

$$r_{\rm b}^i = \left[r_{\rm b}^0 + \sum_{j=0}^{i-1} r_{\rm n}^j \right] \mod (N_{\rm HC}) \ .$$

We will denote a partitioning state symbolically by $\{r_{\rm b}, r_{\rm n}\}$. For the example in Figure 2-5, we have $r_{\rm b}^i = \{11, 14, 38\}$ and $r_{\rm n}^i = \{3, 24, 15\}$.

One should always keep in mind the circular topology of the domain decomposition data structures. The set of Hilbert curve indices is a circle with length $(2^m)^3$. The set of the Hilbert curve raw indices is a circle with length $n_h^0 n_h^1 n_h^2$. The set of partitions is again a circle of length $n_{\rm pr}$.

2.4 Load Balancing

Having introduced the Hilbert curve, we next consider how to use it, that is, how to choose the partitioning state each timestep. We wish to do this so as to balance the workloads of all processes so as to maximize the parallel efficiency. This section discusses details of our dynamic domain decomposition algorithm.

2.4.1 Definitions of Workload, Load Imbalance, and Repartitioning

Repartitioning is the run-time (dynamic) change of particle domain decomposition in order to solve the load balancing problem. Repartitioning is performed by shifting the HC raw indices r_b^i (i.e. the cross-hatched bars on Fig. 2-5) to minimize the load imbalance by minimizing the resulting expected maximum work load per process.

In a discrete time evolution problem like ours, the simulation is synchronized among the processes each timestep, meaning that the amount of time spent by a cluster of computers on a given timestep is given by the maximum amount of wall clock time spent by any process in the cluster doing its share of the problem. We define the *workload* of a process as the wall clock time that it takes for the process to complete one timestep, including the communication waiting time. The amount of wall clock time spent by a process depends on the structure of the workload assignment.

Wall clock time is the number of elementary operations (clock cycles) a processor performs for a given parallel process divided by the CPU frequency. During some of those cycles the processor may be idle or working on other tasks; we call those computationally useless periods waiting time and distinguish them from CPU time. Because the different parallel processes must be synchronized (at several points) each timestep, the workload of each process is given by the wall clock time, and may be decomposed as follows:

$$\text{Wall Clock Time} = \frac{\text{CPU Time}}{\text{Average CPU Usage}} = \text{CPU Time} + \text{Waiting Time} \; . \qquad (2.23)$$

Wall clock time is measured using the system call `ntp_gettime()`.

Ideally, we would like to eliminate the waiting time so that at all times all CPUs are doing useful work. The waiting time has a very complex and non-local structure as it depends on communication and other factors unrelated to the computations done by one process. (For example, on multiprocessor nodes, different processes compete for memory access.) In our treatment, we balance only the CPU time of different processes. Because the wall clock times of all processes are forced to be the same by synchronization, if the CPU time is balanced then there will be no waiting time aside from the minimal amount required for communication and memory access.

The CPU time of a process may be divided into two parts: one that can be attributed entirely to the content of individual HC cells (e.g. particle data) and all the rest (e.g. FFT). The dominant HC cell-specific and CPU-intensive portions of the P^3M code are the PM-density and force interpolation and the PP-force pair

summation. They execute at 100% CPU usage (as they involve no interprocess communication). All the contributions are summed to define the P^{3}M *instantaneous CPU workload* at timestep n for an HC cell at timestep n as

$$\begin{aligned} \tilde{w}^{\mathrm{PM}}(n) &= \text{PM-density and force assignment wall clock time} \\ \tilde{w}^{\mathrm{PP}}(n) &= \text{PP pair summation wall clock time} \\ \tilde{w}^{\mathrm{P3M}}(n) &= \tilde{w}^{\mathrm{PM}}(n) + \tilde{w}^{\mathrm{PP}}(n) \ . \end{aligned} \tag{2.24}$$

We use wall clock time to measure the CPU workload for these portions of the computation because there is (ideally) no waiting time.

Given a set of local *cell workloads* w (which may differ from $\tilde{w}^{\mathrm{P3M}}$) for all the cells $\mathbf{c} \in L_h$ local to each process, we define the *CPU workload of process* i as

$$W_{\mathrm{HC}}(L_h^i, w) \equiv \sum_{\mathbf{c} \in L_h^i} w \ . \tag{2.25}$$

(Note that we use lower case w for the workload of a single HC cell and upper case W for the total workload of all HC cells assigned to one process.) We use a subscript HC because the total CPU time of P^{3}M is dominated by the HC cell-specific PM and PP computations and only these portions of the code need be included in the workload. The other significant cost, the FFT, is automatically load-balanced by FFTW. Note that W_{HC} depends on the local domains and other factors hence it may be varied by repartitioning as discussed below.

The *load imbalance* is defined as a function of the set of all CPU workload W^i on each process as

$$\text{Load Imbalance} \equiv \mathcal{L}(W) \equiv 1 - \frac{\langle W \rangle}{\max\ W^i} \ , \tag{2.26}$$

giving the fraction of time that any processes are waiting instead of computing. The quantity $\langle W \rangle$ is the average of W^i over processes i. In practice, we use $W^i_{\mathrm{HC}}(L_h, w)$ for the workload W^i.

The cell workload defined by equation (2.24) ideally should be proportional to the number of floating point operations needed to compute the relevant parts of the force calculation. However, the measured cell workload (wall clock time) is affected by other factors. For example, there are frequent, unpredictable runtime changes in the efficiency of CPU cache memory management. (Most CPUs have a speed much greater than the memory bandwidth.) In addition, there may be multiple processes running on one (single- or multi-processor) computing node and their competition for system resources affects wall clock time. In addition, if some CPUs in the cluster are slower than others, the workload measurement for the same cell will be higher when measured by the slower processes.

The result of these complications can be large fluctuations in the cell workload measurements that are not repeatable from one timestep to another and therefore interfere with our attempts to load balance. We represent these complications by noting that the instantaneous cell workload defined by equation (2.24) depends on

several factors:

$$\tilde{w}(\text{particle positions, CPU predictable factors, CPU fluctuations}) \ . \tag{2.27}$$

To reduce our sensitivity to unpredictable CPU fluctuations, we introduce *effective cell workloads* as

$$w(n) = \begin{cases} fw(n-1), & \tilde{w}(n) > fw(n-1) \\ (1/f)w(n-1), & \tilde{w}(n) < (1/f)w(n-1) \\ \tilde{w}(n), & \text{otherwise} \ , \end{cases} \tag{2.28}$$

where f is a constant parameter and n is the timestep. The effective cell workload is a time average with clipping to eliminate large fluctuations. It is slightly more accurate than the instantaneous workload for predicting the workload of the next timestep. A series of tests with large simulations showed that the optimal value parameter is $f \approx 2.0$.

The *instantaneous* and *effective load imbalance* are defined by equation (2.26) using equations (2.24) and (2.28) respectively for the cell workloads. The instantaneous load imbalance represents the fraction of time that the parallel processes spend idle, while the effective load imbalance is an estimate of the same fraction in the absence of CPU fluctuations.

Each timestep n, we compute the values of instantaneous $\mathcal{L}^n_{\text{ins}}$ and effective $\mathcal{L}^n_{\text{eff}}$ load imbalance. We perform repartitioning each time when the value of the effective load imbalance exceeds the maximum tolerance value. The *target partitioning state* $\{r'_{\text{b}}, r'_{\text{n}}\}$ (see § 2.3.4) should be chosen so as to minimize the expected value of the instantaneous load imbalance during the force evaluation next timestep. Aside from the target partitioning state, that value also depends on the unknown cell workloads at the next timestep. To find the optimal partitioning state, one may estimate the cell workload in the next timestep very well using its latest measured value

$$w(n+1) \approx w(n) \ . \tag{2.29}$$

As illustrated by equation (2.27), the cell workload during the next timestep is a function of the unknown particle positions at the next timestep. However, since particles do not move far in one timestep compared to the size of a HC cell, we can ignore this dependence for now. The other two arguments factors determining the cell workload are due mainly to the effectiveness of CPU cache memory management, which depends on the memory layout and is hard to predict. The main change in the memory layout during the next timestep is a different partitioning state which means different local regions. By introducing the technique described in §2.5.1, we eliminate the dependence of the second argument in equation (2.27) on local region assignment. The third argument of equation (2.27) can not be eliminated and is the main cause of inaccuracy of equation (2.29), as demonstrated in §§2.7.3 and 2.7.5 using test simulations.

The *residual load imbalance* is defined as the minimum possible load imbalance, computed with equations (2.25) and (2.26) allowing for arbitrary repartitioning, based

on the effective cell workloads of the current timestep:

$$\mathcal{L}_{\rm res}(W') = \min_{\{r'_{\rm b}, r'_{\rm n}\}} \mathcal{L}(W) \ . \tag{2.30}$$

We seek to find the partitioning state that minimizes $\mathcal{L}_{\rm res}(W')$, called the *target partitioning state.* With this choice of partitioning, $\mathcal{L}_{\rm res}(W')$ will become an estimate for the effective load imbalance of the next timestep.

Even in the absence of CPU fluctuations, the residual load imbalance cannot be reduced to zero because of the granularity of the workload distribution across HC cells. For an extremely clustered matter distribution, the workload $w_{\rm max}$ of the densest HC cell within the simulation volume may be greater than the average workload of all processes, $w_{\rm max} \geq \langle W \rangle$. (This requires extreme inhomogeneity because most processes have thousands or even millions of HC cells associated with them, while the slowest to finish may have only one HC cell.) The granularity of the HC method requires that each process have at least one HC cell. In this case, the residual load imbalance is bounded by

$$\mathcal{L}_{\rm res} \geq 1 - \frac{\langle W \rangle}{w_{\rm max}} \ . \tag{2.31}$$

In this regime there is no point in extending the problem to a larger number of processes, since the wall clock time will be given by that of the process holding the cell $w_{\rm max}$ (§ 2.7.5). In general, the N-body problem is scalable only up to a number of processes given by

$$n_{\rm pr} \leq \frac{W_{\rm tot}}{w_{\rm max}} \ . \tag{2.32}$$

Improved load balance can be achieved by further subdividing the computation of short-range forces using an adaptive mesh refinement technique, as we will demonstrate in a later chapter.

2.4.2 Repartitioning and Memory Balancing

As discussed in §2.3.4 the local regions at any given time are completely specified by the current partitioning state $\{r_{\rm b}, r_{\rm n}\}$. The target partitioning state is given by a primed set $\{r'_{\rm b}, r'_{\rm n}\}$. The target partitioning state can be reached from the initial one by a sequence of sets of $n_{\rm pr}$ non-overlapping *elementary partition shifts* $\Delta r^i_{\rm b}$ along the circle indexed with the HC raw indices, so that

$$r'^i_{\rm b} = r^i_{\rm b} + \sum_{i=0}^{n_{\rm pr}-1} \Delta r^i_{\rm b} \ .$$

It is efficient to perform each set of the elementary partition shifts in two stages: first by moving simultaneously all the even partitions followed by the movement of all the odd ones. This way, during each of the two stages, the entire process group will decouple into pairs of adjacent processes each involved with an elementary partition shift exchanging particles with the other process in the pair.

Given the initial and target partitioning states, each partition can be moved from its starting to its target state in one of two possible directions along the circle. We define a parametric isomorphic linear mapping R_b that takes the initial partitioning state $\{r_\mathrm{b}, r_\mathrm{n}\}$ into the target one $\{r'_\mathrm{b}, r'_\mathrm{n}\}$ as the parameter α goes from zero to one:

$$\begin{aligned} R_b^0(\alpha) &\equiv r_\mathrm{b}^0 + \alpha \left[(r_\mathrm{b}'^{,0} - r_\mathrm{b}^0) \mod (N_\mathrm{HC}) \right] , \\ R_n^i(\alpha) &\equiv r_\mathrm{n}^i + \alpha [\, r_\mathrm{n}'^i - r_\mathrm{n}^i \,] , \end{aligned} \tag{2.33}$$

where N_HC is the total number of HC cells and the partition $i = 0$ is treated so as to ensure a circular topology. It follows that

$$R_b^i(\alpha) = R_b^0(\alpha) + \sum_{i=0}^{j-1} R_n^j(\alpha) . \tag{2.34}$$

The initial and target partition state starting indices are given by $r_b^i = R_b^i(0) \mod (N_\mathrm{HC})$ and $r_b'^i = R_b^i(1) \mod (N_\mathrm{HC})$, respectively. The direction of movement of the individual partitions along the circle in our code is given by differentiating equation (2.34) with respect to α.

The target partitioning state is reached from the initial one by the sequence of maximal non-overlapping elementary partition shifts in the directions specified by the above procedure until the target partitioning state is achieved. All of the partition-dependent data are adjusted to reflect the change of partitioning state. The corresponding particle sends and receives are performed and the relevant cell data are exchanged. In addition, the irregular particle domains are reallocated for each process participating in any of the resulting elementary partition shifts.

In order to avoid paging one needs to impose a total memory constraint for repartitioning. Since the memory associated with particles dominates the problem, while doing repartitioning we check whether the reallocation of the particle array on the receiving processes succeeds. If it does not, we divide the requested number of cells $|\Delta r_\mathrm{b}^i|$ by two and try the repartitioning again. This procedure guarantees that we satisfy the memory limit on each process.

Another practical consideration arises when using a cluster with multi-processor or multi-process nodes. As a result of Hilbert curve domain decomposition the memory loads and cache usage of sequential processes are correlated. These correlations can make it more difficult to achieve load balance. One should therefore avoid assigning sequential processes to the same computational node.

2.4.3 Finding the Optimal Target Partitioning State

In this section, we show how to find the target partitioning state $\{r'_\mathrm{b}, r'_\mathrm{n}\}$ that minimizes load imbalance (eq. 2.26), given the current HC cell workloads and the current partitioning state $\{r_\mathrm{b}, r_\mathrm{n}\}$. As discussed in §2.4.1, we assume that the current cell workloads are an adequate predictor of those at the next timestep, equation (2.29).

Cell Workload Data Compression

The optimal target partitioning state depends on the workloads of every HC cell on every process, $w(j)$ for $j \in [0, N_{\rm HC})$. This information can be represented as a one-dimensional *continuous total workload bar* of length $W_{\rm tot}$ equalling the total work summed over all cells. For each HC cell we mark the bar with vertical dashes at positions

$$u(r) = \sum_{j=0}^{r} w(j), \quad r = 0, \ldots, N_{\rm HC} - 1 \ , \tag{2.35}$$

which gives the cumulative workload of cells up to the one with raw index r. Figure 2-6 illustrates this with continuous total workload bars C_0 and C_1. The horizontal spacings between the adjacent dashes (the white stripes) represent the cell workloads of each cell: $w(r) = u(r+1) - u(r)$. Each white stripe is due to the cell workload associated with one cell. A single dash however may be an overlap of thousands of very close dashes showing up as one due to the limited resolution of the figure.

In a large N-body simulation, the total number of HC cells is huge. For example, in the simulation described in §2.7.2, $N_{\rm HC} = 2.36 \times 10^7$, which requires $N_{\rm HC} \times$ `sizeof(float)` = 94.5MB to hold the values of the workloads. This memory requirement grows with the volume of the simulation box and if the mesh is large enough the problem of finding the optimal partitioning state is impossible to process serially (i.e. on one of the cluster nodes).

To solve this problem we compress the cell workload data by discretizing it. The total workload bar is divided into $N_{\rm bin}$ segments per process, or $M_{\rm bin} = n_{\rm pr} N_{\rm bin}$ segments in total. The continuous total workload array $u(r)$ is replaced the much smaller array $B(k)$ with $k \in [0, M_{\rm bin})$. Figure 2-6 illustrates this with the bars D_0, D_1, and D_2. Each array member $B(k)$ is assigned to the subinterval $[k\,\Delta W, (k+1)\Delta W)$ of the total workload bar, where $\Delta W \equiv W_{\rm tot}/M_{\rm bin}$. The value $B(k)$ is defined as the number of cell boundaries (the dashes) within the corresponding subinterval of the total workload bar. The non-zero members $B(k) > 0$ correspond to the filled rectangles of bars D_0–D_2 in Figure 2-6.

Suppose we start from the initial partitioning state $\{r_{\rm b}, r_{\rm n}\}$ marked by triangles above C_0 in Figure 2-6. We define a *discrete partitioning state* $\{\hat{r}_b, \hat{r}_n\}$ in the discrete workload space by $\hat{r}_b^i \equiv [u(r_{\rm b}^i)/\Delta W]$, $0 \le i < n_{\rm pr}$, where the square brackets signify taking the integer part; $\hat{r}_n^i$ is the spacing between the consecutive $\hat{r}_b^i$ along the binned bar of length $M_{\rm bin}$. We define the workloads in the discretized problem as $\hat{W} \equiv \hat{r}_n^i$. Following equation (2.26), the load imbalance of a discrete partitioning state is defined by

$$\hat{\mathcal{L}}[\hat{r}_n] = 1 - \frac{\langle \hat{r}_n \rangle}{\max \hat{r}_n} \ . \tag{2.36}$$

The residual load imbalance is redefined in the discrete space as [cf. eq. (2.30)]

$$\hat{\mathcal{L}}_{\rm res}[\hat{r}'_n] = \min_{\{\hat{r}'_b, \hat{r}'_n\}:\, B(\hat{r}'_b) > 0} \hat{\mathcal{L}}[\hat{r}'_n] \ . \tag{2.37}$$

The problem of load balancing is posed in the discrete space as finding the discrete

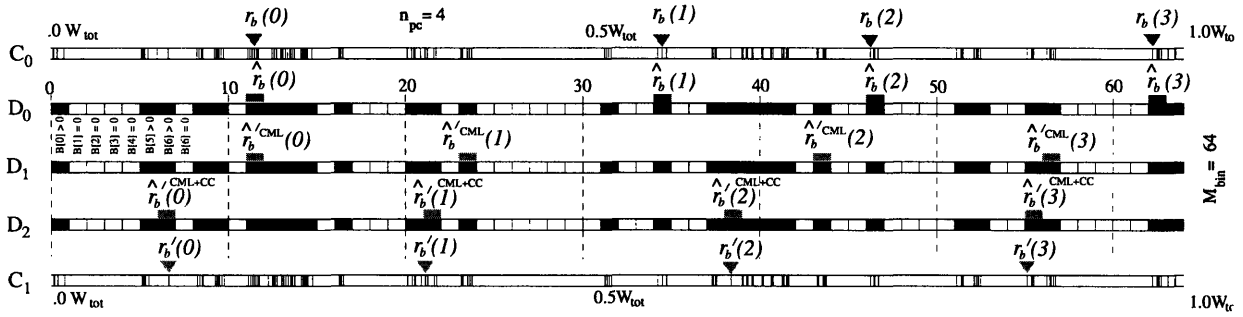

Figure 2-6: Representation of the HC cell workloads using continuous (C_0 and C_1) and discrete (D_0, D_1, and D_2) workload bars, as described in the text. This example is for a simulation on $n_{\rm pr} = 4$ processes, with HC-mesh of size $n_h^0 = n_h^1 = n_h^2 = 100$ and $N_{\rm bin} = 16$. C_0 and C_1 are the continuous total workload bars before and after repartitioning, respectively. The filled triangles give the locations of the initial (C_0) and target (C_1) partitions. A bin $B(k)$ along the bar D_0 is filled if and only if the number of dashes in the same interval of bar C_0 is non-zero. The discrete partitioning states are marked by the filled rectangles above the filled bins of bars D_0–D_2. The solution in the discrete space marked on bar D_2 is obtained by first repartitioning $D_0 \rightarrow D_1$ [holding $r_b(0)$ fixed] and then shifting $D_1 \rightarrow D_2$. Finally, the continuous target partitioning state $\{\hat{r}'_b, \hat{r}'_n\}$ marked on bar C_1 follows from D_2. Note that the topology of each bar is a circle formed by connecting its ends.

target partitioning state $\{\hat{r}'_b, \hat{r}'_n\}$ that will minimize the load imbalance. We discuss how this is done in the next subsection.

Once the discrete target partitioning state $\{\hat{r}'_b, \hat{r}'_n\}$ is found, the continuous target partitioning state $\{r'_{\rm b}, r'_{\rm n}\}$ is also found by setting $r'^i_{\rm b} = r^i$, where r^i is the raw HC index of any cell such that $u(r^i) \in [\hat{r}'^i_b \Delta W, (\hat{r}'^i_b + 1)\Delta W]$. There are, in general, many HC indices that will accomplish this. For example, in Figure 2-6, the final triangles for bar C_1 may be placed at any dash lying beneath the rectangles above bar D_2. The choice is arbitrary and this freedom in setting the target partitioning state will result in negligible differences in the residual load imbalance $\leq 2/M_{\rm bin}$. In practice, we set the partition at the first HC cell that lies in the desired interval.

Finding the Target Partitioning State in the Discrete Case

There are two practical approaches to solving the discrete target partitioning state problem of equation (2.37).

In the *cumulative repartitioning* approach we keep the zeroth partition fixed while setting the other ones as close as possible to being equally spaced along the discrete workload bar, subject to the constraints $B(\hat{r}'^i_b) \neq 0$. It is evident that the resulting target partitioning state is a function of only the initial position of the zeroth partition $\hat{r}^0_b$ and the discrete workload array B^k, $k \in [0, M_{\rm bin})$.

The cumulative approach alone is not satisfactory for optimizing the discrete

load imbalance equation (2.36) when the cell workloads of some of the HC cells far exceed the discretization load $w(j) \gg \Delta W,\ j \in [0, N_{\rm HC})$. Indeed this problem is illustrated in Figure 2-6. The initial discrete partitioning state is given by $\hat{r}_b = \{11, 34, 46, 62\}$ as shown by the rectangles above the workload bar D_0. Applying the cumulative approach using the above rule, we have $\hat{r'}_b^{\rm CML} = \{11, 23, 43, 56\}$, and $\hat{r'}_n^{\rm CML} = \{12, 20, 13, 19\}$, yielding load imbalance $\mathcal{L}^{\rm CML} = 1 - (16/20) = 0.2$, which is relatively poor. (The superscript CML is used for partitions found with cumulative repartitioning.) This approach uses only the position of the zeroth partition and the discrete cumulative workload array. It is insensitive to differences in the adjacent workloads, e.g. $\hat{r}_n^i$ and $\hat{r}_n^{i+1}$.

In the *circular cyclic correction repartitioning* approach (denoted by superscript CC), we start from a partition i and shift it to the bin $\hat{r}_b^i = k$ such that it is the closest possible distance to the bin in the middle of the two adjacent partitions, $k = ([\hat{r}_b^{i-1} + \hat{r}_b^{i+1}]/2$. After the correction of the partition i is done, we move on to the next partition $i+1$, applying the same technique but using the already corrected value for the position of partition i. We then continue applying the same scheme for all the other partitions in cycles along the circle $i \in [0, n_{\rm pr})$ until the resulting shifts for all partitions $i \in [0, n_{\rm pr})$ become zero. The resulting positions of the partitions will define the target state in the circular cyclic correction repartitioning approach. This approach if used alone is not satisfactory just as for the cumulative partitioning approach above, however the nature of the problem is completely different. If a large variation in workload $\hat{r}_n^i$ develops across a large range of indices i (e.g. between i and $i + n_{\rm pr}/d$), this variation will not be suppressed by the circular cyclic correction scheme since only the adjacent partitions $\hat{r}_b^{i-1}$ and $\hat{r}_b^{i+1}$ are used for correction of any given partition $\hat{r}_b^i$. On the other hand, all the local fluctuations in workload will be suppressed very effectively.

As we see, the cumulative repartitioning approach and the cyclic circular partitioning approaches smooth the large scale and small scale (in terms of the range of indices) workload fluctuations respectively. Applying the two approaches in sequence works well to provide a nearly optimal solution for the discrete workload. In the example of Figure 2-6, the bar D_2 shows the result of applying the circular partition correction approach to the output of the cumulative approach (bar D_1) obtained from the initial discrete partitioning state (bar D_0). As follows from the bar D_2 of Figure 2-6, the resulting target partitioning state is $\hat{r}_b^i = \{6, 21, 38, 55\}$ and $\hat{r}_n^i = \{15, 17, 17, 15\}$. The resulting discrete load imbalance is $\hat{\mathcal{L}} = 1 - (16/17) = 0.06$ is 3.4 times smaller than the load imbalance obtained using only the cumulative method. Our experiments show that the combination of the two approaches results in a good approximation to the load-balanced target partitioning state. The residual load imbalance is generally limited not by our ability to find the optimal solution but instead by the CPU time fluctuations due to variations in cache usage.

2.5 Particle Data Layout and Communication

In a serial code, the array of particle structures (2.10) is static, that is, it remains fixed length with unchanging particle labels. In a parallel code with domain decomposition, particles may move from one process to another. This not only requires interprocessor communication, it also complicates the storage of particle data. This section discusses our solutions to these problems.

2.5.1 Linked List Structure, Particle Movement, and Sorting

The particle data are stored as a single local particle array of pointer $[\texttt{pa}, \texttt{pa\_f})$ on each process. A slightly larger range $[\texttt{pa}, \texttt{pa\_fa})$ is allocated to avoid reallocation every timestep. In addition to the particle array, we have a linked list that tells which particles lie in each HC cell. For each HC cell there is a pointer (the root) that (if it is non-null) points into the particle array to the first particle in that HC cell. A complete list of particles within a given local HC region L_h^i is obtained by dereferencing the appropriate linked list root and then following the linked list from one particle to the next, as illustrated in Figure 2-7. The linked list also has a root `hc_avb` that points to disabled particles.

There are several challenges associated with this simple linked list method of particle access. First, one must transfer particles between processes. Second, HC cells are themselves exchanged between processes as a result of repartitioning. Third, one must optimize the traversal of the linked lists to optimize code performance. Finally, one must specify which HC cells are associated with a given process. We discuss these issues in the remainder of this section.

During each position advancement equation (2.11), twice every timestep some particles move across the boundary of their local particle domain. As a result, such a particle is sent from a process i to another process j whose local region L_h^j it entered. Particles may cross the boundary of any pair of domains. The associated communication cost scales linearly with the L_h surface area. The Hilbert curve domain decomposition minimizes this cost because of the low surface to volume ratio (§ 2.3.2).

When a particle `p` moves outside the local region L_h^i, it leaves a gap in the local particle array. We set the particle mass to -1 and call this particle array member a disabled particle. All the disabled particles on each process form a separate linked list with root `hc_avb`. The particles entering L_h^i from other processes replace the disabled particles or are added to the end of the particle array.

As a particle initially in process i crosses a boundary to another process, the id of the target process j should be immediately found in order to send this particle to the new process. Dividing the new particle coordinates by the HC mesh spacing gives the new Hilbert curve mesh cell coordinates $\mathbf{c}$. The target process id can then be found calling Moore's function for the new HC index $h = \mathcal{H}(\mathbf{c})$. By using the current Hilbert curve partitioning, one finds the id of the target process j from h. Once all particles to move have been identified, the particles are transferred between processes.

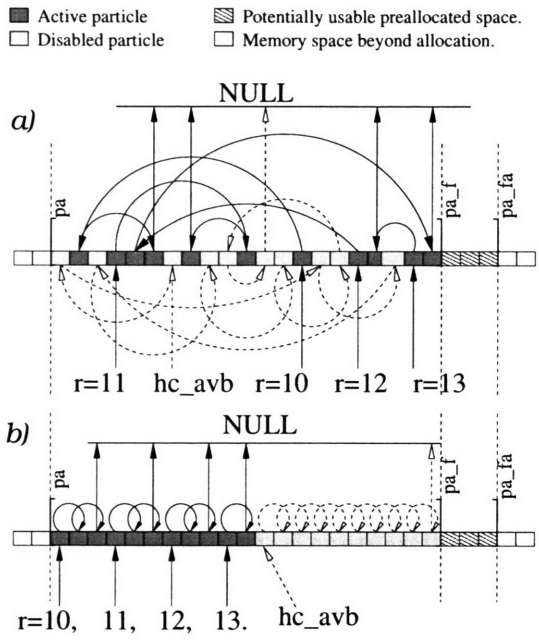

Figure 2-7: Particle array structure and access in the parallel code. This example corresponds to process $i = 0$ of Fig. 2-5. The HC cells associated with this process are $r = (10, 11, 12, 13)$. The particle arrays are the horizontal bars (with disabled particles corresponding to gaps in the array opened up when particles moved to other processes). The linked list is given by the arrows going from one particle to another; the solid (dashed) arrows give the linked list for the active (disabled) particles. The linked list roots are the pointers `hc_avb` (for disabled particles) and $r = (10, 11, 12, 13)$ (for active particles) beneath the particle array bars. Each linked list begins at a root and ends with the `NULL` pointer. The particle array is allocated slightly more storage (`pa_fa`) than needed (`pa_fa`). a) The particle array and linked list before sorting. b) The same particles and the linked list after sorting.

As we show in Appendix 2.10, Moore's function calls are relatively expensive. To avoid having this cost each time a particle crosses the boundary, we allocate an extra one layer of HC cells surrounding the boundary of L_h^i, as shown in Figure 2-8, and we mark the surrounding cells with the ids of the appropriate processes j by calling Moore's function for each of them exactly once. By doing this once, we avoid calling Moore's functions in the future. However we still have to call the function for the very small fraction of the boundary-crossing particles that went further than one boundary layer cell in one timestep. The extra layer of HC cells surrounding the local region is also used with the particle-particle force computation as described in §2.6.3.

We maintain the particle linked list throughout the simulation instead of reforming it each timestep. As particles cross from one HC cell to another — even if they are in the same local region L_h^i — the linked list is updated to reflect these changes. The particle array is reallocated whenever the fraction of disabled particles exceeds a few percent (the exact value is a parameter set by the user), or the amount of particles

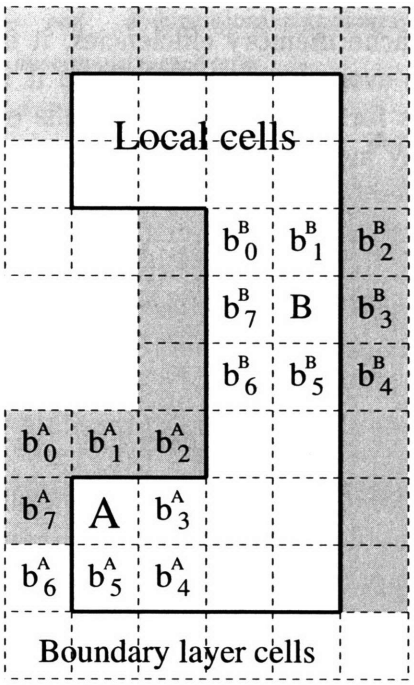

Figure 2-8: Hilbert curve mesh cells. The cells within the solid line are the L_h^i cells containing all the particles assigned to process i. Information about the layer of boundary cells (all gray and white cells outside the local region) is also stored by process i. This information is used both when particles are transferred between processes and during the short-range (particle-particle) force computation. In the latter case, the particle data for the shaded cells is used to compute forces on particles in cells A and B as discussed in § 2.6.3.

exceeds the boundary of the pre-allocated particle array `pa_fa`.

In addition to the pointer to the root of the linked list that contains all the particles within each HC cell, each cell of the local region contains other structure members: the process number the cell belongs to, the current and previous timestep cell workloads required by equation (2.28), the number of particles in this cell, etc. We will refer to this structure as the *HC cell structure* and the array of structures for all HC cells the *HC cell array*. One member of this array has size 16 bytes. When repartitioning occurs, we send and receive the relevant HC cell array members and the particles they contain to the appropriate processes.

Some program components, such as particle position advancement, require access to the complete particle list on each process. All local particles can be accessed using the particle array and filtering out the passive particle array members as follows:

```
for(p = pa; p < pa_f; p++){
  if(p->mass == -1.) continue;
  ...
}.
```
(2.38)

We found that because of cache memory efficiencies, it is up to ten times faster to use a simple array to access every local particle than it is to dereference the three-dimensional linked list roots for each of the local cells of L_h^i. The reason for such difference is that simple array members are sequential in the machine memory, while the successive linked list members are not, and the CPU cache memory is more effectively used when data are accessed sequentially in an array. The improvement in efficiency is especially important in the particle-particle calculation because each particle is accessed many times during one force computation.

Here we introduce a fast sorting technique that places the particle data belonging to the same HC cell sequentially within the segments of the particle array, ordered by increasing HC-cell raw index. This sorting procedure is performed each timestep before the force computation.

Every timestep, before a force calculation, we follow all the L_h cells in the order of their raw HC index, and concatenate their linked lists, resulting in just one linked list of all the particles in the local particle array. Then, using the unnecessary acceleration `g0` and `g1` members of the particle structure as pointers, we form an extended linked list replacing the old one. The result is a new linked list which can be traversed both forward (using `g1`) and backward (using `g0`). Then, starting from the first particle of a simple array of particles, we swap it with the first particle in the extended linked list while the forward and backward pointers of the immediately adjacent within the extended linked list particles being updated. We then proceed to the next particle in the simple array and in the linked list doing the same, until we have sorted the entire particle list. The result of this sorting is illustrated by Figure 2-7b.

In addition to optimizing the CPU cache memory usage, the above sorting technique eliminates the need to allocate an additional buffer for sending and receiving particles while repartitioning, because all the particles to be moved as the result of repartitioning will occupy contiguous segments in the simple particle array. When the sorting is completed the original linked list is unnecessary and is deallocated in order to be formed again directly using the sorted particle array, before the particle advancement and repartitioning take place.

To transfer particles between processes we use a modification of `MPI_Alltoallv` that assures no failure will occur if insufficient memory has been pre-allocated for the send and receive buffers. This achieved by using `MPI_Alltoall` to exchange the numbers of particles to be sent and received and then using as many `MPI_Alltoall` and `MPI_Alltoallv` calls as necessary to avoid overflowing the available memory of each processor.

2.5.2 Scalable Allocation Local Region Access

As mentioned above, during particle exchange and force computation one needs frequent access to a cell's particle list and other cell data, given the indices $\mathbf{c}$ of the cell in the HC mesh. The most obvious method is to call Moore's function $h = \mathcal{H}(\mathbf{c})$ to get the global HC index and then use our table of HC entries (§ 2.3.3) to convert h into the raw index r. The raw index then gives the root to the particle linked list as shown in Figure 2-7. This method is unsatisfactory because of the expense of calling

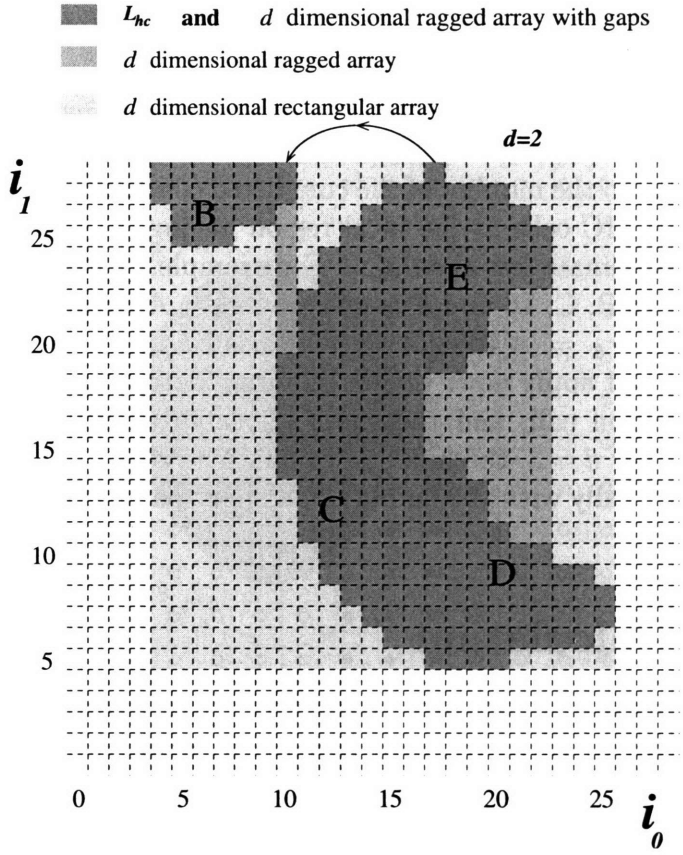

Figure 2-9: Schematic illustration of the HC local region L_h (dark gray) assigned to one process and the rectangular array that includes it (light gray combined with dark gray). The ragged array (middle gray combined with dark gray) requires much less storage but only the ragged array with gaps (dark gray) corresponds exactly to L_h. Four cells belonging to L_h are randomly selected and labelled B, C, D and E.

Moore's function many times during the force evaluation.

Another simple method of allocation for the L_h cells would be a *d-dimensional rectangular array* of cells holding the frequently used roots of the linked lists to the particles contained in this cell and the total number of particles within it. The access to a HC cell given its coordinates $\mathbf{c}$ in this case is given by dereferencing the array $r = A[c_0][c_1][c_2]$ in the case of $d = 3$, where A is an array of HC cell raw indices (or pointers to HC cells) updated after each repartitioning. The problem here, illustrated in Figure 2-9, is that many of the entries of A are wasted because the HC local regions are not rectangular parallelpipeds. This can be improved by adjusting the bounds of the array indices (c_0, c_1, c_2) to the extremal values for cells in the local region. The result is a simple *d-dimensional ragged array*, also illustrated in Figure 2-9.

The optimal method of local region HC cell allocation and access is to add one more dimension to the array of HC cell pointers A used in a simple ragged array. The extra dimension accounts for variable number of disjoint parts in the last dimension. This method allocates the minimal storage needed beyond the number of HC cells in

L_h^i. We call this a *d-dimensional ragged array with gaps.* The HC cell is then obtained by dereferencing the $(d+1)$-dimensional array A.

To access a cell with coordinates $c_0 \ldots c_{d-1}$ using a d-dimensional ragged array with gaps, we use $r = A[c_0][c_1] \ldots [c_{d-2}][\mathcal{M}][c_{d-1}]$, where $\mathcal{M} = \mathcal{M}(c_0 \ldots c_{d-1})$ is the integer function equal to the number of the completed contiguous intervals in the c_{d-1}-ordered set of all the HC cells in the local region having coordinates $c_0 \ldots c_{d-2}$ and having $(d-1)$-th coordinate less than c_{d-1}. For example, in the case $d = 2$ of Figure 2-9, access to the cells B, C, D, and E is given by $r_B = A[6][0][26]$, $r_C = A[12][0][12]$, $r_D = A[20][0][9]$ and $r_E = A[18][1][23]$. The disadvantages of the other methods considered above do not apply now: the $(d+1)$-array dereference call is exponentially faster than the function call, and the space allocated exactly equals the required number of L_h cells. For $d = 3$, the function evaluation $\mathcal{M}(c_0, c_1, c_2)$ takes a time that grows only logarithmically with the number of disjoint parts along the last dimension for a give c_0 and c_1.

2.6 Force Calculation

In this section, we present an efficient method for parallel PM and PP computation of forces for particles within the HC local regions. By using the techniques developed in §§2.4 and 2.5, we have made our algorithms load balanced and efficient.

2.6.1 PM Force Calculation

The PM force calculation requires communication between two different data structures with completely different distributions across the processes. The particles on one process are organized into irregularly-shaped HC local regions. The density and force meshes, on the other hand, have a one-dimensional slab decomposition based on FFTW. The parallel computation is an SPMD implementation of the five PM steps presented in §2.2.5.

Definitions

We define a few concepts that will be needed in order to describe and implement the data exchange between the two different data structures during the parallel PM force calculation. The various sets used in the calculation are illustrated in Figure 2-10.

The FFTW parallel Fast Fourier Transform implementation [22] allows one to compute forward and inverse Fourier transforms of the complete three dimensional array of $n_0 n_1 n_2$ mesh points distributed among the processes j in the form of slabs of $\texttt{nloc}(j)\, n_1 n_2$ grid points, where $\sum_j \texttt{nloc}(j) = n_0$, each slab starting at the position $\texttt{sloc}(j) = \sum_{i=0}^{j-1} \texttt{nloc}(i)$ along the 0-th dimension. We will call these slabs the *density* or *force mesh slabs* (depending on the context) and denote them by G_{sl}^j. The geometry of the slab G_{sl}^j is calculated once and for all at the start of the run by calling the FFTW Fourier transform plan initialization routine.

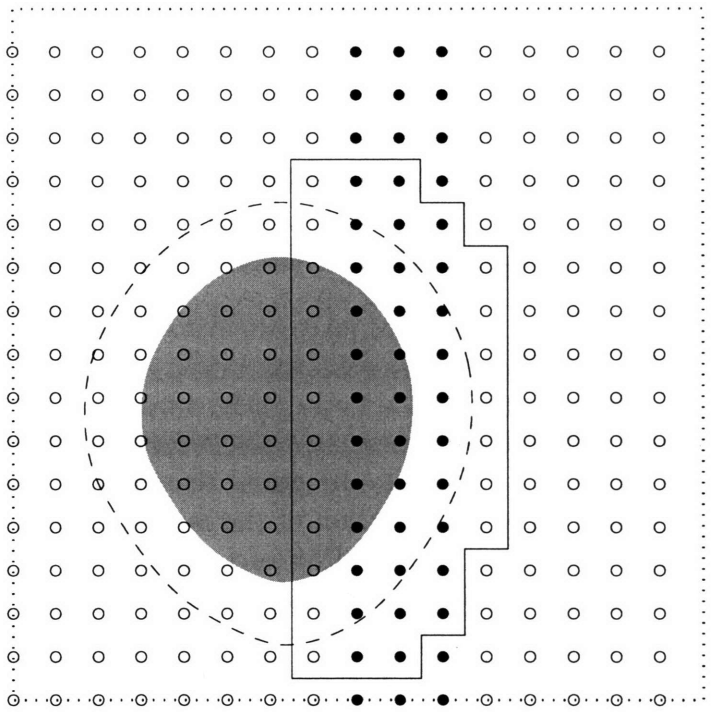

Figure 2-10: Schematic representation of the sets used in the PM force calculation. The volume within the dotted line is the total simulation volume V_0, the small circles are the discrete set of PM density gridpoints G_0. The filled circles are the PM gridpoints within a FFTW slab j, $G^j_{\rm sl}$. The gray filled region is the HC local region L^i_h. The set of all circles within the dashed line is $\mathcal{G}(L^i_h)$; the set of filled circles within the dashed line is $G^j_{\rm sl} \cap \mathcal{G}(L^i_h)$. Extending this last set slightly gives the continuous set within the solid line, $\mathcal{L}(G^j_{\rm sl} \cap \mathcal{G}(L^i_h))$. Eq. (2.41) gives the intersection of this last set with the gray regionL^i_h.

Let us denote the complete discrete set of all density mesh gridpoints needed for a complete Fourier transform by G_0, and the complete continuous set of all positions within the whole simulation volume by V_0. We have

$$\begin{aligned} G^j_{\rm sl} \in G_0, \quad & \bigcup_j G^j_{\rm sl} = G_0 \\ L^i_h \in V_0, \quad & \bigcup_i L^i_h = V_0 \ . \end{aligned} \tag{2.39}$$

Here, i labels the process holding the HC local region while j labels the process holding a given density/force mesh slab.

For a continuous set of positions $L \in V_0$, let us define $\mathcal{G}(L)$ to be the minimal complete subset of the density grid points $\mathbf{X}_{\rm gr} \in G_0$ such that equation (2.13) is satisfied for any position vector $\mathbf{X} \in L$. By this definition, if all the local particles are contained within L, after the density assignment of Step 1 of the PM force calculation, the only non-zero PM-density grid points of G_0 are in fact within a subset $\mathcal{G}(L) \in G_0$.

For a discrete subset $G \in G_0$ of the density gridpoints, let us define $\mathcal{L}(G)$ to be

the minimal complete continuous set of points $\mathbf{X} \in V_0$ such that equation (2.13) is satisfied for any $\mathbf{X}_{\rm gr} \in G$. Now, if all the grid points local to a process are within a subset $G \in G_0$ of all the particles in the simulation volume V_0, only the particles of the subset $\mathcal{L}(G)$ may acquire any non-zero force contribution from those gridpoints during *Step 5* of the PM-force calculation.

Optimal PM Communication Strategy

As we discussed in §2.2.5, Step 1 of the PM force calculation involves filling the density grid points in $G^j_{\rm sl} \in G_0$ using the particles distributed in the volumes $L^i_h \in V_0$. Steps 2–4 involve working only with $G^j_{\rm sl}$ and are straightforward since they do not require any interprocessor communication aside from the parallel FFT. During Step 5 the information flows in the exactly opposite direction, therefore an algorithm for Step 1 applies to Step 5 as well with the direction of the information flow reversed. The problem remaining now is for Step 1 of the PM force calculation to decide how to fill the local density grids $G^j_{\rm sl}$ from the particles distributed within the local regions L^i_h. To solve this problem we considered a number of approaches described briefly below, but only the last one is implemented in our code and is effective over the entire range of clustering.

a) Sending Particles. Under this method, each pair ij of processes sends the appropriate portion of the particle data from process i to process j to fill the density mesh $G^j_{\rm sl}$ of slab j. For each pair of processes the set of the density gridpoints

$$G^j_{\rm sl} \cap \mathcal{G}\left(L^i_h\right) \tag{2.40}$$

on process j will be updated with the particles brought from the volume

$$L^i_h \cap \mathcal{L}\left(G^j_{\rm sl} \cap \mathcal{G}\left(L^i_h\right)\right) \tag{2.41}$$

within the HC local region of process i.

This method is very efficient for the pairs where the particle sender processes i have low particle number density, thus reducing the number of particles to be sent and the communication cost.

b) Sending Grid Points. Under this method, each pair ij of the processes fills the portion (2.40) of the grid points using the local particles within (2.41), then sends the filled gridpoints to process j.

This method performs poorly when the particle number density is low on the sender process, because most of the density values in the message are zero. This method is very efficient for the pairs where the particle sender processes i have a high particle number density: each gridpoint of the sender process contains the contributions from many particles.

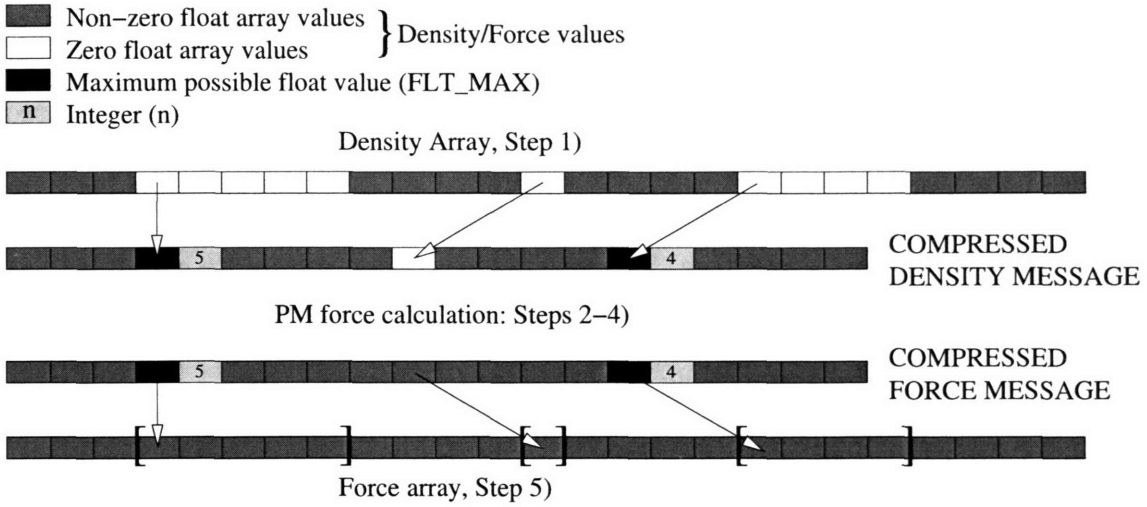

Figure 2-11: Sparse array compression of density and force messages during PM force computation. The two density arrays (top two bars) are equivalent but the lower one is compressed by run-length encoding. Compression is signalled by a special data value (`FLT_MAX`) followed by the number of zeros. The compressed array on process i is sent to process j using an MPI function call. A compressed force message is constructed on process j using the template given by the density message. The forces are sent back to process i and expanded. The bracketed values in the bottom array can be ignored because there are no particles nearby the relevant grid points.

c) Combined Particle and Grid Point Send. Method *a)* is effective with low particle number density while method *b)* is effective with high particle number density on the particle sender process. The idea of the combined particle and grid point send method is to choose for each pair of processes the approach that requires sending the least data.

d) Sending Compressed Grid Points. This approach optimizes the communication cost in both the extreme cases of low and high number density of the particles on the sender process i. The idea behind this method is to use the approach *b)* above and apply *sparse compression* to the gridpoint messages (2.40). As we know, the grid point approach performs poorly when the particle number density is low on the sender process. Using sparse compression as we explain in the following subsection significantly alleviates this problem by reducing the message size for the underdense regions L_h^i.

Sparse Compression of Grid Point Messages

In a cosmological simulation, the overdense regions have small HC local regions with every grid point having many nearby particles so that the force and density messages are small. On the other hand, low-density regions have large HC local regions with

many PM grid points but the density and force messages are made small by the compression method illustrated in Figure 2-11.

During Step 1 of the PM computation, if a number of binary zeros are encountered in the grid message, they all are substituted by a pair of numbers before sending packets: the first number is a delimiter (an illegal density or force value such as `FLT_MAX`) and the second number is an integer giving the number of zeros to follow in the original uncompressed message. This technique is called *run-length encoding.* The resulting compression factor is unlimited and depends on how frequent and contiguous the zero values are positioned in the grid message. The receiver process j simply uncompresses the message by filling the gridpoints within $G^j_{\rm sl} \cap \mathcal{G}(L^i_h)$.

During Step 5, the force values are sent from process j to i three times (once for each of the three dimensions). The force array message is identical in the size to the density message that was sent during Step 1 for each pair ij of processes. We compressed the density values in Step 1 using run-length encoding of zero value densities. In the force message the technique runs into a difficulty because the gravitational forces are long range forces by nature and their values are nowhere equal to zero. If we do not compress the force values, there is no advantage in choosing the compressed gridpoint approach, since the force messages would have the same length as the uncompressed density messages.

By using packet information obtained while receiving the density array, we can compress the forces using exactly the same pattern formed by the packets of the density message, as shown in Figure 2-11. The receiving process will decompress the force and obtain exactly the initial force array excluding the values of force at the array members which were skipped in the density assignment (the square bracketed force values in Fig. 2-11). This loss of information is however completely irrelevant for interpolation of the force values to the particles in *Step 5* because the square bracketed force values in the force array belong to grid points which earlier acquired absolutely no density values from the surrounding particles, which means that for that grid point and for any particle within L^i_h, the gridpoint has no nearby particles [the condition (2.13) is not satisfied]. Thus the force values at that grid point will not be interpolated to any particles during Step 5.

The idea of sparse array compression is not implemented in the Hydra code [37]. Once implemented it will significantly reduce their communication and memory costs.

2.6.2 Practical PM Implementation

Equation (2.40) gives the minimal set of density grid points on process j needing to be filled with values from particles on process i. This set is impractical to work with because of its irregular shape. For a practical implementation we embed this region within a rectangular submesh of G_0 during Steps 1 and 5 of the PM force computation, as follows.

For a continuous set of positions inside the simulation volume $L \in V_0$, let us define $\mathcal{R}(L)$ to be the minimal rectangular subset of density grid points such that $\mathcal{G}(L) \in \mathcal{R}(L)$. For grid points with $\mathcal{R}(L)$ but outside $\mathcal{G}(L)$ we set the density values

to zero. It follows at once that if we use

$$G_{\rm sl}^{j} \cap \mathcal{R}\left(L_h^i\right)$$

instead of equation (2.40) for the definition of PM grid point messages, we will have the rectangular mesh $\mathcal{R}\left(L_h^i\right)$ for interpolation of density for particles within L_h^i, and this still give the correct result. However, since the extent of the local region L_h^i inside the simulation box is not limited, neither is the extent of $\mathcal{R}\left(L_h^i\right)$. For example, when L_h^i consists of just two cells with the coordinates $(0,0,0)$ and $(n_h^0-1, n_h^1-1, n_h^2-1)$, it is easy to see from the definition that $\mathcal{R}\left(L_h^i\right)$ encloses the whole simulation density mesh G_0 as a subset and this is too much memory space for allocation on a process.

To avoid this problem, we dissect the local region L_h^i uniformly into n_k^i slices M^{ik} along the 0-th dimension so that the extent of each slice along the 0-th dimension will not exceed $n^0/n_{\rm pr}$. Using the previous equation we have, now summed for all the receiving processes $j = 0 \ldots n_{\rm pr}-1$

$$\sum_{j=0}^{n_{\rm pr}-1} G_{\rm sl}^{j} \cap \mathcal{R}\left(L_h^i\right) = \sum_{j=0}^{n_{\rm pr}-1} G_{\rm sl}^{j} \cap \mathcal{R}\Big(\sum_{k=0}^{n_k^i-1} M_h^{ik}\Big) = \sum_{k=0}^{n_k^i-1} \left(\sum_{j=0}^{n_{\rm pr}-1} G_{\rm sl}^{j} \cap \mathcal{R}(M_h^{ik})\right) \quad . \quad (2.42)$$

For each slice M_h^{ik} of the HC local region L_h^i, the density is interpolated onto the rectangular mesh $\mathcal{R}(M_h^{ik})$ which is small enough to be allocated since its extent in the 0-th dimension is limited by roughly $n^0/n_{\rm pr}$ grid points. Then, the messages under the inner sum of equation (2.42) are sent to processes $j = 0..n_{\rm pr}-1$. The procedure is repeated for each slice M_h^{ik}.

In the code presented in this chapter we use the blocking MPI routines for PM message communication, which requires synchronization between each pair of processes exchanging the message. In order to reduce waiting time, MPI allows bi-directional blocking communication using `MPI_SendRecv`. In the above equation the process i is described as the *sender* of the PM-grid messages obtained by interpolation from the particles within L_h^i to the processes j in order to update their FFTW-slabs $G_{\rm sl}^j$. Note however, that the same process i also behaves as a *receiver* of the PM grid messages from the other processes j in order to update the FFTW-slab $G_{\rm sl}^i$. The set of the received messages is obtained by simply swapping the indices i and j in the above equation. Adding the two together we have for the set of gridpoints participating in the communication on process i in both directions

$$\sum_{j=0}^{n_{\rm pr}-1} \left[G_{\rm sl}^{j} \cap \mathcal{R}\left(L_h^i\right) + G_{\rm sl}^{j} \cap \mathcal{R}\left(L_h^i\right)\right] = \sum_{k=0}^{n_k^{ij}-1} \sum_{j=0}^{n_{\rm pr}-1} \left[\, G_{\rm sl}^{j} \cap \mathcal{R}(M_h^{ik}) + G_{\rm sl}^{i} \cap \mathcal{R}(M_h^{jk}) \right] \,, \quad (2.43)$$

where $n_k^{ij} \equiv \max(n_k^i, n_k^j)$ and the M_h^{ik} is defined to be an empty set for for $k \geq n_k^i$.

In order to access particles in a given slice M_h^{ik} of the local region we use the particle access technique described §2.5.2. The sorting technique described in §2.5.1 speeds up the density and force interpolation. The timing of the interpolation for each HC cell gives the PM part of the HC-cell workloads in equation (2.24).

The above procedure is used for both density and force interpolation in the PM force calculation. In the current implementation, the MPI messages are blocking, which means additional waiting time. In a subsequent chapter we describe the implementation of non-blocking communication resulting in a significant speedup of the PM calculation.

2.6.3 PP Force Calculation

The particle-particle (PP) force calculation increments forces acting on each of the particles in a pair if the particles are closer than $\tilde{R}_{\max}$.

The method of particle access developed in §2.5.2 allows one to access all the particles within a given HC cell. From equation (2.19), HC cells are coincident with the chaining mesh cells needed for the PP force calculation. To see how the communication and computation work, consider the example of Figure 2-8. To compute the PP force for a particle p within chaining mesh cell A, the particle data in the surrounding cells $b_0^A \ldots b_3^A$ are required. The particle data within the cell b_3^A are locally available. However one needs to bring the positions and masses from the other processes to get the particle data for the boundary layer cells $b_0^A \ldots b_2^A$. Once the particle data from the boundary layer cells are gathered, the PP force calculation may be performed by pair summation, after which the resulting forces for the particles within $b_0^A \ldots b_2^A$ are sent back to their processes where the PP forces of the original particles are incremented.

The same algorithm applies to any other cell within L_h, for example the cell B of Figure 2-8, for which the particle data for b_0^B and b_1^B are available locally while the particle data for cells b_2^B and b_3^B must be brought to the local process from the others. Because of its pairwise nature only half the surrounding cells are needed for the PP force calculation for each HC cell. In total, the particle data for the non-local cells shaded in Figure 2-8 are required for the PP force calculation for each particle within L_h. The amount of communication needed for a complete PP force calculation is proportional to the number of particles in the cells required to be brought from the other processes through the boundary layer cells.

If the PP pair summation step is started synchronously on all processes, it will finish at approximately the same time on all processes if the load imbalance is low. Otherwise, the processes that complete the PP force computation first will have to wait for the remaining processes to finish their pair summation. Since the pair summation is the most time-consuming step of P^3M, it is crucial that the procedure be load-balanced. This is accomplished using the methods of §2.4. The CPU time of the pair summation step is used in the cell workload calculation of equation (2.24). The particle access time in the pair summation loop is minimized by pre-sorting the particles as described in §2.5.1.

2.6.4 Memory Management

In early versions of our code, the memory often exceeded the available resources causing the code to crash. By implementing runtime tracking of memory usage we were able to identify the problems and optimize the memory requirements. Memory

Table 2.3: Dominant memory requirements of the parallel `llp3m-hc` code. Here $n(i)$ is the number of particles and $\texttt{nloc}(i)$ is the thickness of the PM slab , both on process i.

	Notation	Memory Size, per process i	Total Memory size
Particle array	$M_{\rm P}$	4 bytes $\times$ $11N(i)$	4 bytes $\times$ $11N$
Particle linked list	$M_{\rm L}$	4 bytes $\times$ $N(i)$	4 bytes $\times$ N
HC mesh	$M_{\rm HC}$	4 bytes $\times$ $5r_{\rm n}^i$	4 bytes $\times$ $5n_h^0 n_h^1 n_h^2$
Green's function	$M_{\rm G}$	4 bytes $\times$ $\texttt{nloc}(i)\, n^1(n^2/2+1)$	4 bytes $\times$ $n^0 n^1(n^2/2+1)$
Density and Force meshes	$M_{\rm PM}$	4 bytes $\times$ $2\texttt{nloc}(i)\, n^1(n^2+2)$	4 bytes $\times$ $2n^0 n^1(n^2+2)$
FFTW scratch space	$M_{\rm FFT}$	4 bytes $\times$ $\texttt{nloc}(i)\, n^1(n^2+2)$	4 bytes $\times$ $n^0 n^1(n^2+2)$
PP boundary layer particles	$M_{\rm PP}$	4 bytes $\times$ $8\Delta n_{\rm PP}$	4 bytes $\times$ $8\sum \Delta n_{\rm PP}$

usage was reduced largely in three ways: the irregular shaped local domain memory technique of §2.5.2, the elimination of particle buffer allocation while repartitioning, and memory balancing when necessary during repartitioning as described in §2.4.2.

In Table 2.3 we list the main memory requirements for our `llp3m-hc` code. Compared with the memory requirements for the serial code in Table 2.1, we were able to reduce the size of the particle linked list by 50%. Note that the HC mesh is the parallel equivalent of the serial chaining mesh but requires 5 times more storage. (This is still less than the storage required for the linked list, so we have accomplished a net savings.) The FFTW scratch space is optional but significantly improves FFTW performance, so we allocate it. The maximum memory allocated per process each timestep can be obtained by combining the tabulated values in the sequence that follows their actual allocation and release in the code. For example, the memory spaces $M_{\rm PM}$, $M_{\rm FFT}$ and $M_{\rm PP}$ are allocated when the memory space $M_{\rm L}$ is released. The actual memory usage along with the detailed measurements from a 800^3 run are described in §2.7.3.

2.7 Tests

The first and most important test was to verify that our parallel PM and P^{3}M codes give identical results to the serial codes (both the original Fortran codes and their C translations) when given identical inputs, to within the precision of machine roundoff error. The serial codes have been thoroughly tested by [28].

The remaining tests presented in this section test the performance of the parallel codes in order to optimize the performance. Several of the innovations described in the preceding sections were devised in response to performance tests.

The PM test presented in §2.7.1 was performed on a beowulf cluster consisting of 7 nodes each with a 1.7 GHz Pentiun-4 processor with 256 KB L2 cache memory, 1 GB RAM memory, and 34 GB of hard drive, connected by 100 Mb/s ethernet. The Linux `gcc` compiler was used. This cluster has a Linpack performance of 15 GFlops.

The rest of the test runs were performed on a beowulf cluster consisting of 20 dual Xeon 2.4 GHz Pentium-4 and 512 KB L2 cache memory processor computing

nodes each containing 4 GB RAM memory and 360GB of disk, connected by gigabit Ethernet. The Intel `icc` compiler was used. This cluster has a Linpack perfomance of 70 GFLops.

2.7.1 PM Simulation of Extremely Clustered Matter

Cosmological initial conditions for cold dark matter were generated for a simulation with 512^3 particles and grid points, with a power spectrum having a Gaussian cutoff at a wavelength equal to one-fourth of the box size and white noise on larger scales. The resulting nonlinear evolution, shown in Figure 2-12, leads to the formation of two massive particle clusters displaying many phase space caustics. The initial conditions were evolved using both the `llpm-sl` and `llp3m-hc` codes to compare their timing performance.

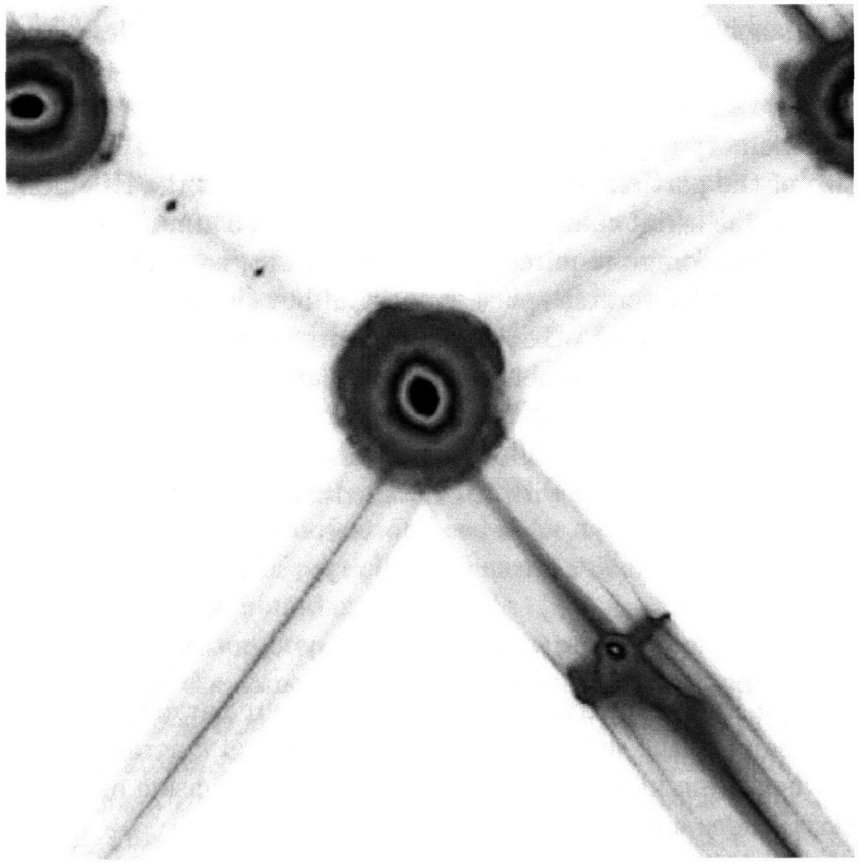

Figure 2-12: Projected particle distribution for the entire PM simulation volume of 512^3 particles at timestep 5693. False colors scale with the logarithm of projected mass density. Strong clustering like this favors dynamic rather than static domain decomposition methods.

This test used an early version of `llp3m-hc` with only PM forces. Since we did not compute PP forces the constraint given by equation (2.19) was not in effect.

Instead we set our HC mesh spacing to $\Delta\tilde{x}_h^i \geq 1.5$. The sorting technique described in §2.5.1 was not implemented. Instead of equation (2.24), we used the number of particles in a cell to define the HC cell workload. Repartitioning therefore resulted in an approximately equal number of particles on all processes at each timestep.

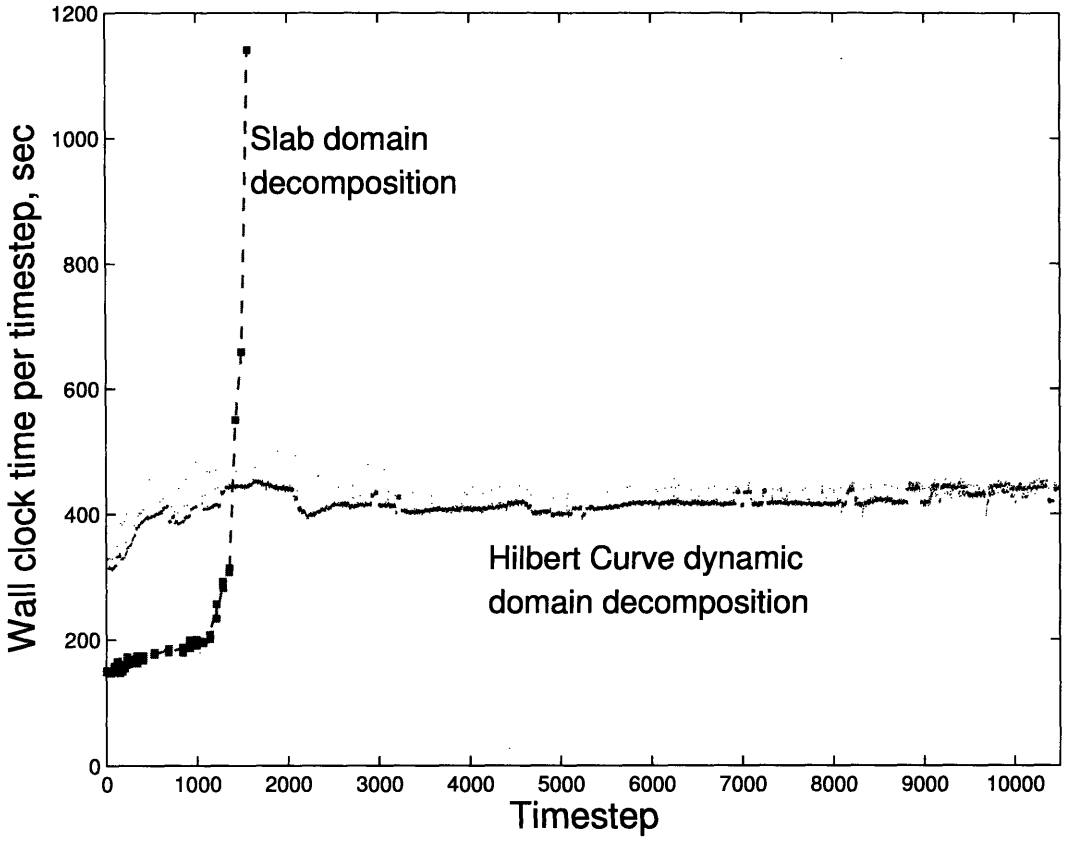

Figure 2-13: Timing performance comparison of a Hilbert curve dynamic domain decomposition code `llp3m-hc` and a fixed (slab) domain decomposition parallel code `llpm-sl` for the identical run showed in Fig. 2-12. The runs start from the linear regime and are evolved using only PM forces.

Figure 2-13 shows the wall clock time per timestep for the Hilbert curve code `llp3m-hc` and the fixed slab domain decomposition code `llpm-sl`. As we see from the Figure, the HC-based PM code evolves very far into the regime of strong matter clustering without any significant slowdown. On the other hand, the slab decomposition code grinds to a halt because of the growing memory imbalance arising in any fixed domain decomposition method. As more and more particles end up on one process, not only does its CPU workload grow, but the process eventually runs out of memory and starts paging to disk, slowing down the evolution by orders of magnitude. Only a dynamic domain decomposition can handle clustering as extreme as that shown in Figure 2-12.

Even though the HC code is vastly superior to slab decomposition under strong

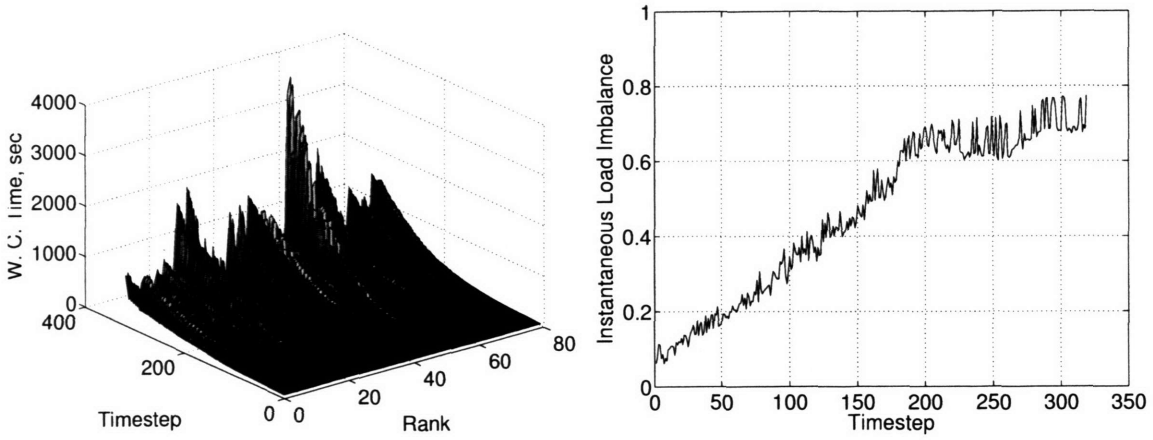

Figure 2-14: Left: Time required to complete the pair summation step on each process (labelled by MPI rank) as a function of timestep during a 800^3 P^{3}M run without repartitioning. Right: Instantaneous load imbalance as a function of timestep for the same run.

clustering, it is slower at early times. This is mainly because the local regions are displaced from the FFTW slabs in the `llp3m-hc` code, therefore more communication is required. In addition, since the non-blocking communication was not implemented (see § 2.6.2), there is some unnecessary waiting time in the HC code.

2.7.2 P^{3}M Simulation of ΛCDM without Repartitioning

An extensive series of tests were performed using the `llp3m-hc` code to assess its behavior under a wide range of clustering conditions. All of the runs use one particle per PM mesh cell in a cube of size $L^0 = L^1 = L^2 = 200$ Mpc. The Plummer softening length was set to $\tilde{\epsilon} = 0.4$ (i.e. 40% of the PM mesh spacing). We generated the initial conditions for the ΛCDM model (with $\Omega_m = 0.27$, $\Omega_\Lambda = 0.73$, $H_0 = 71$ km s^{-1} Mpc^{-1}, $\sigma_8 = 0.84$, $n = 0.93$ from Bennett et al. 2003) using the BBKS transfer function in a C parallel version of `grafic1` [7]. The timestep parameter of equation (2.12) was set to $\eta_t = 0.05$.

As a first test of the full `llp3m-hc` code we ran a simulation with 800^3 particles and grid points with no repartitioning. This run was performed with 80 processes on 20 nodes (40 CPUs using hyperthreading, which treats a physical CPU as two virtual CPUs with improved performance). Without repartitioning the HC local regions on each process remain the same throughout the run. The results appearing in Figure 2-14 are predictable. A few processes require much longer time to finish the pair summation, leading to a large load imbalance. Late in the simulation, only about $1 - \mathcal{L} = 25\%$ of the net wall clock time is spent doing computation; most of processes sit idle most of the time waiting for the heavily loaded processes to finish the PP pair summation.

2.7.3 P^{3}M Simulation of ΛCDM with Repartitioning: Load Balancing

We reran the 800^3 simulation of §2.7.2 on 80 processes with repartitioning enabled in order to load balance the computation. Because of the strong increase in clustering and the resulting growth of the PP pair summation time, the wall clock time to complete one timestep increased from just over 4 minutes at the beginning of the simulation to 2 hours at the end (timestep 569, when the expansion factor was $a = 0.7$, or a redshift of $z = 0.43$). The simulation took two weeks to get to this point and would have required another month to evolve to $a = 1$ provided it remained well load balanced. In a subsequent chapter we introduce an adaptive technique that substantially decreases the PP workload enabling longer and more highly clustered simulations to be performed in much less time. A projection of the particle distribution at timestep 566 for this simulation was used in Figures 2-3 and 2-4. At the end of the simulation the Layzer-Irvine energy conservation check (eq. 2.9) was satisfied to a precision $\tilde{E}_{\rm con}/\tilde{E}_g = 5 \times 10^{-5}$. (Energy conservation can be as much as 100 times worse when clustering is very weak and the PM force contributions dominate over PP.)

In Figure 2-15, we present the instantaneous and residual load imbalance as functions of timestep. The effective load imbalance (not shown) is almost identical to the instantaneous load imbalance but is lower by 10% for the highest spike at timestep 403. The differences between these various measures of load imbalance are explained in §2.4.1.

One of the main results of this chapter is that the time-averaged instantaneous load imbalance generally remains between 10 and 15% (averaging 12%) and does not grow steadily worse with time. By contrast, without Hilbert curve dynamic domain decomposition, by timestep 300 the load imbalance exceeded 70% (Fig. 2-14).

At early times the residual load imbalance is much less than the instantaneous load imbalance because fluctuations in cache usage limit our ability to predict the optimal repartitioning for the next timestep. Later in the run, the residual load imbalance grows when a small number of highly-occupied HC cells begin to dominate the CPU time in the PP force calculation, as we discuss further below.

To analyze the limitations of our load balancing algorithm, in Figure 2-16 we plot the instantaneous workload measured every timestep for every process using equations (2.24), (2.25), and (2.28). The workload is approximately the CPU time required for PP pair summation summed over all the local HC cells on each process. The main pattern seen is the steady rise of the average PP workload with timestep due to the increase in clustering caused by gravity. After that, one sees in the left plot four spikes in processes 14, 34, 54, and 74. Given our assignment of processes to nodes, these processes reside on the same physical node 14 and are probably caused by competition of these processes for memory access. Fluctuations in cache memory usage are probably also responsible for the smaller fluctuations in workload superposed on the steady rise with timestep in the right plot.

To investigate further the cause of growing residual load imbalance, in Figure 2-17 we present the discrete workload array (§ 2.4.3) at timestep 568. A perfectly load-

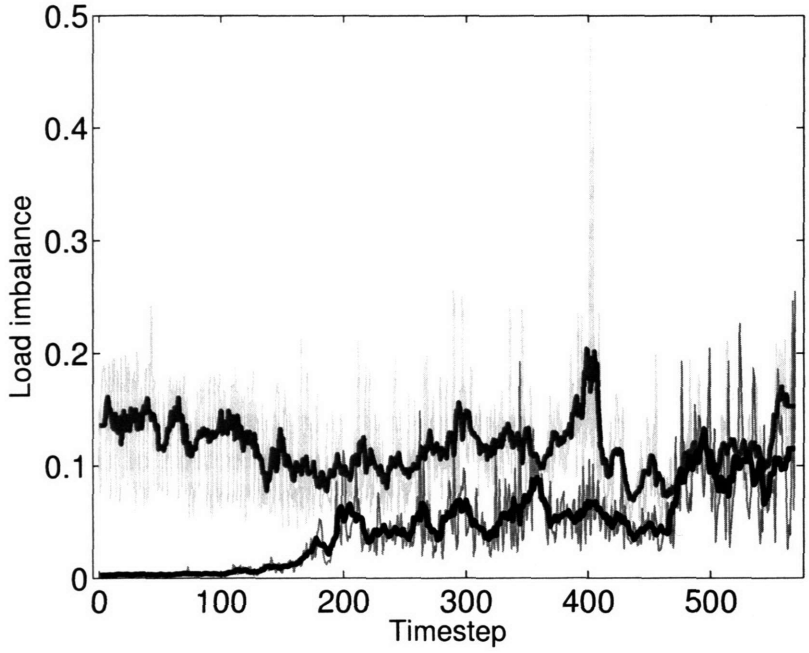

Figure 2-15: Instantaneous (blue and yellow curves) and residual (black and pink curves) load imbalance as a function of timestep for the 800^3 P^3M run with Hilbert curve repartitioning on 80 processes. The yellow and pink curves give the load imbalance of every timestep; the blue and black curves apply boxcar averaging over 10 timesteps to reduce the fluctuations. The residual load imbalance is the minimum possible load imbalance that could be achieved by repartitioning in the absence of fluctuations.

balanced partitioning state would correspond to a target partitioning state with all boundaries lined up in one vertical column. (In that way, a fraction 1/80 of the work would be assigned to each of the 80 processes.) However, this is impossible because processor boundaries cannot occur in white sections (where by definition there are no processor boundaries) and every column contains some white space. Instead, the (nearly) optimal solution is found using the method described in §2.4.3 and used to define the target partitioning state corresponding to the black numbers giving the process boundary for each process.

Figure 2-17 shows that the workloads of processes 48–53 are hard to adjust by repartitioning since most of the cells of the workload array in their vicinity are white. This occurs because these processes have a small number of HC cells in very high density regions requiring a significant amount of CPU time to complete their PP force calculations. The resulting uneven workload assignment causes the systematic increase in the measured instantaneous workloads for these processes after timestep 470 in Figure 2-16 and therefore an increase in the residual load imbalance in Figure 2-15. We would not be able to carry out the P^3M calculation much further before a single HC cell begins to take more time than the local regions of other processes,

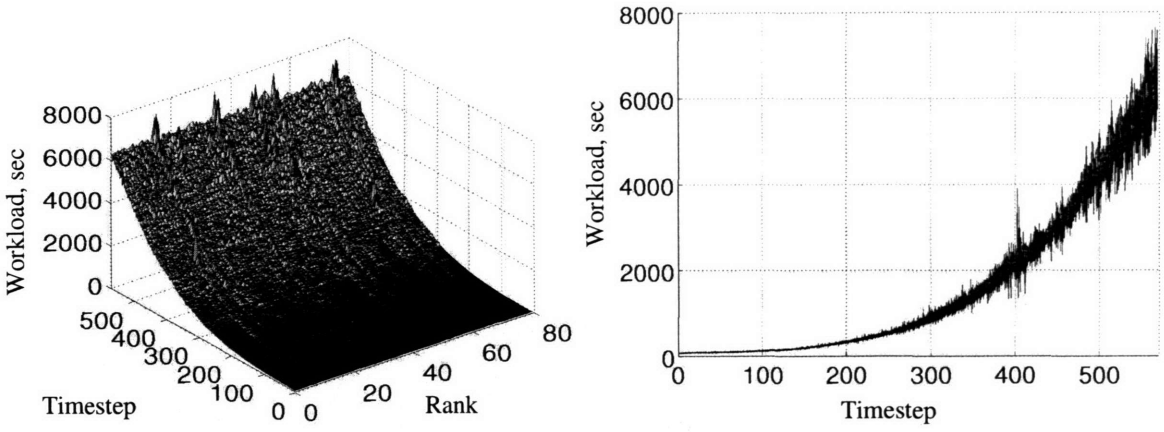

Figure 2-16: Instantaneous workload of each process as a function of timestep. Left: view from an oblique angle. Right: projected view down the rank (process number) axis.

leading to severe load imbalance (eq. 2.31). Even with the strong variation in workload present after timestep 500, Figure 2-15 shows that our algorithm manages to achieve an instantaneous load imbalance almost as small as the optimal (residual) imbalance. We expect even better performance when (in a later chapter) adaptive mesh refinement is used to alleviate the PP pair summation workload.

2.7.4 P^{3}M Simulation of ΛCDM with Repartitioning: Local Regions, Timing, and Memory Usage

We continue discussing the same long 800^3 P^{3}M simulation as in the preceding section but now focus on aspects of code performance other than load balance.

In Figure 2-18 we plot the volume $r_{\rm n}^i$ of HC local regions as a function of timestep and process number. The local region volume is large when the workload per HC cell is small (i.e., in low-density regions) and is small when the workload is high (i.e., in dense particle clusters). At the beginning of the simulation, all 287^3 HC cells are uniformly divided between the processes. As clustering grows, a huge range of volumes develops as the local regions adjust to follow the change in their workload. Because of the compactness property of the Hilbert curve, particle clusters or voids tend to occupy adjacent processes.

At timestep 568 (the end of the run), the smallest local regions belong to processes $i = 20$, 34, 48, 49 and 54, with $r_{\rm n}^i = 7$, 282, 4, 29, and 4 cells, respectively (see also their workload structure in Fig. 2-17). Because the run is well load balanced, the small number of local cells per process implies that the workloads of those cells greatly exceed the average. Indeed, the average value of workload per cell in the whole simulation box is $W_{\rm tot}/(n_h^0 n_h^1 n_h^2) = 4.23 \times 10^{-8}\, W_{\rm tot}$. A process with only 4 local cells has (on average) workload $W_{\rm tot}/(4n_{\rm pr}) = 3 \times 10^{-3} W_{\rm tot}$, which is five orders of magnitude higher than the average workload of a cell in the simulation volume. Such a high cell workload results from the huge local number density of particles for

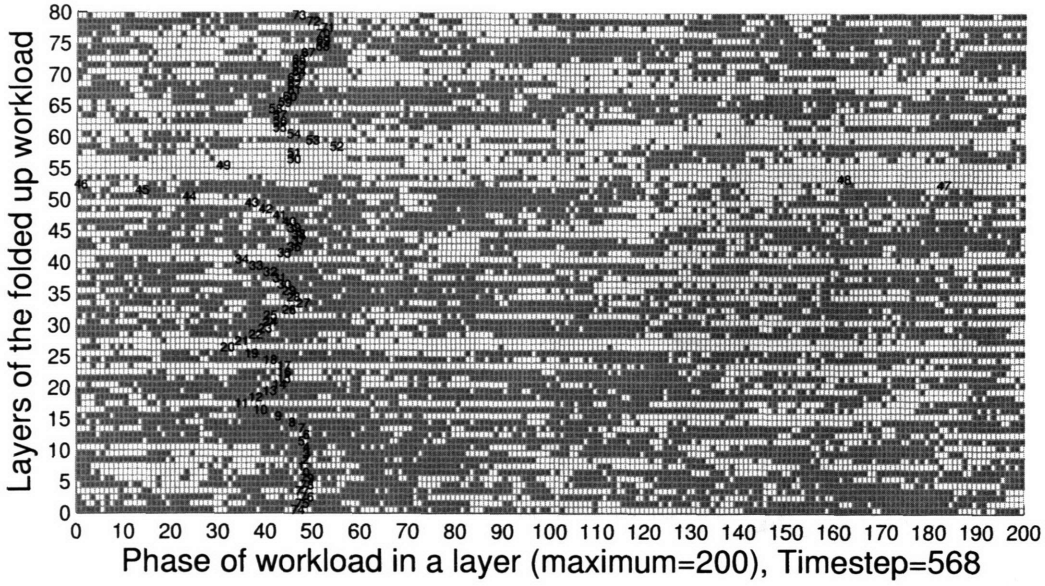

Figure 2-17: The discrete workload array for timestep 568 of the same simulation analyzed in Figs. 2-15 and 2-16. The discrete workload array is a one-dimensional array of $M_{\rm bin} = 1600$ cells folded into a set of 80 layers (along the vertical axis) of length 200 (along the horizontal axis). Blue cells contain at least one boundary between HC cells; a continuous white segment represents a single HC cell. The target discrete partitions $\hat{r'}_b^i$ are marked with process rank i (cf. bar D_2 in Fig. 2-6).

these cells leading to a heavy PP-force calculation load (eq. 2.24). Indeed, process 48 holds 1.9×10^5 particles or 8.8×10^3 times the average. Also, by an unfortunate coincidence, two of the five most heavily loaded processes (34 and 54) ended up on the same compute node, leading to heavy demands on memory and (apparently) causing the the spikes seen in Figure 2-16.

Next we present a detailed analysis of the timing structure of the force calculation.

In Figure 2-19 we present the structure of the wall clock and CPU time of the PM force computation. We see that the wall clock time on average exceeds the CPU time by a factor of 7 during the run, which means that during the PM force calculation processes spend 85% of their time waiting for interprocessor communication requests to clear. This is a big fraction that can be reduced significantly by the use of non-blocking requests for PM density and force grid sends and receives between the local HC and FFTW slabs (see Section 2.6.1). However, except at the early stages of clustering (the initial timestep was 4 minutes, growing to 2 hours), the PM time is a a small fraction of the total timestep.

In Figure 2-20 we present the wall clock and CPU timing for the parallel PP force computation. Three tasks are required for a PP step (§ 2.6.3). First, particle positions and masses must be brought from boundary layer cells on other processes. Second, pairwise gravitational accelerations are computed by direct summation. Finally, accelerations are returned across the domain boundaries as needed. The cost

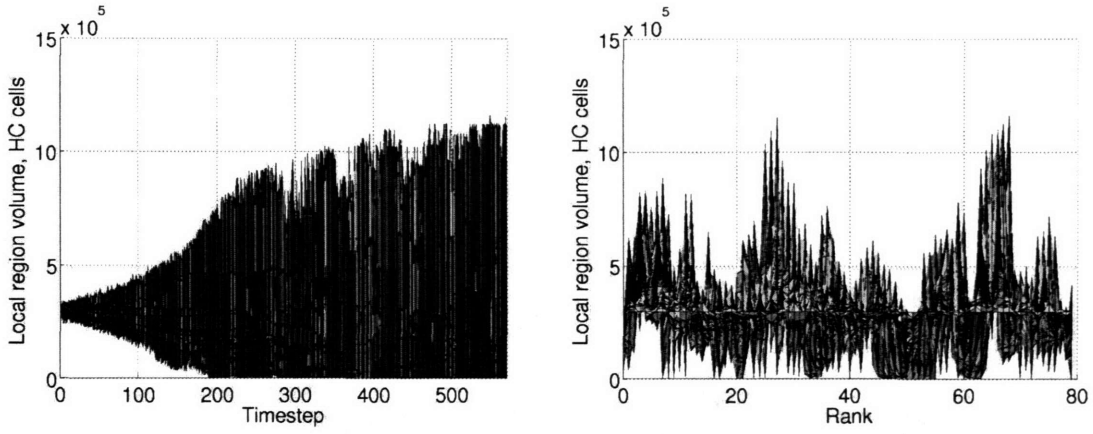

Figure 2-18: Volume of HC local regions, as a function of timestep and process rank. In the left panel, the full range of process ranks is shown for each timestep. In the right panel, the full range of timesteps is shown for each process rank. The average volume of local regions is 3×10^5 cells.

of pair summation dominates the other tasks and has essentially equal wall clock and CPU times (hence involves almost no waiting). For the communication tasks (sending and receiving particle positions, masses, and accelerations) the wall clock time greatly exceeds CPU time because of blocking communication and the resulting waiting time.

Figure 2-20 shows that the waiting time during PP is dominated by the return of accelerations after their computation. The wall clock time of this task is about 10% that of the pairwise computation. This is because the load imbalance in the code arises almost entirely during the pairwise force computation. Figure 2-15 shows that the average instantaneous load imbalance is approximately 12% during the whole run. From equation (2.26), we see this means that one of the processes requires about 12% more time to complete its pair summation than the others (since the total CPU time is dominated by pair summation). Other processes cannot get all their accelerations returned until this process finishes computing them, which explains the order of magnitude difference in the direct summation and return of acceleration wall clock times.

The P^3M force calculation accounts for nearly all the time of each timestep. Some time is spent by repartitioning every timestep. Because repartitioning may result in the exchange of many HC cells and particles between processes, we might expect it to take a significant amount of time. In fact, the total wall clock time spent on load balancing (analyzing workloads, finding the target partitioning state, and exchanging data between processes) takes on average less than 20 seconds per timestep, or less than the CPU time of the PM force calculation. Occasionally the repartitioning time spikes up to nearly 80 seconds but it does not grow steadily with clustering. The load balancing time is generally less than 8% of the total wall clock time per timestep. Less than one percent of the wall clock time is spent updating particle positions and

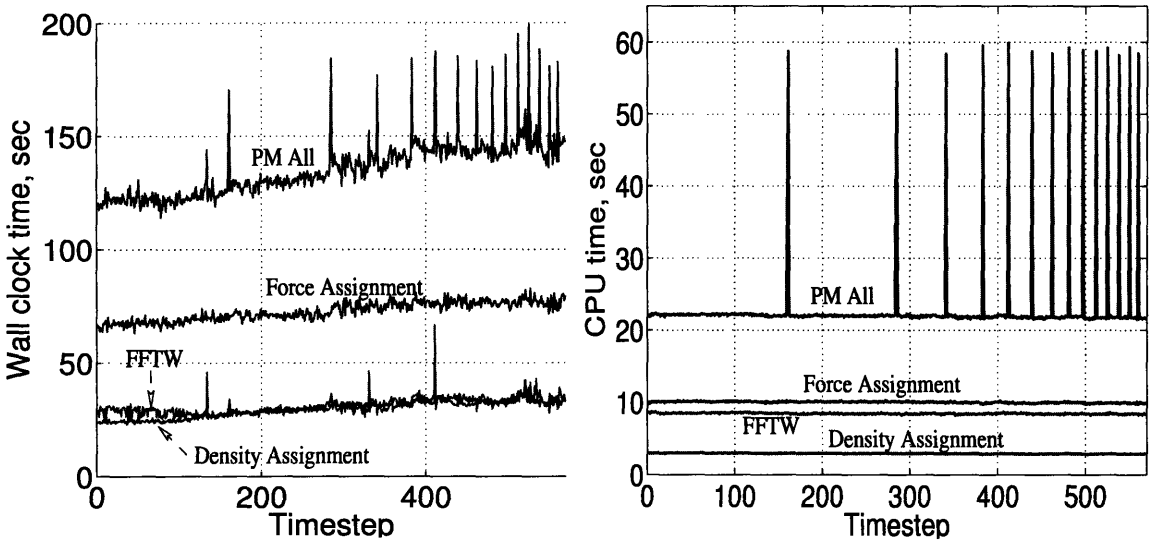

Figure 2-19: Structure of the wall clock time (left) and CPU time (right) of the PM force computation. The times are averaged over processes. The 40-second spikes are due to recomputation of the PM Green's function when the code is restarted every 24 hours. Unlike the wall clock time, the CPU time does not increase because it does not include time spent waiting for processes to finish.

velocities and exchanging the particles between processes as a result of their motion.

Next we analyze memory usage during the run. In §2.6.4 we estimated the memory requirements. We now compare these estimates with measured memory usage.

In Figure 2-21 we present the maximum amount of total memory allocated by a given process during any timestep within the run. In Linux, a memory request in excess of about 1.4 GB on any process will crash the run (see § 2.2.7). As expected, at the beginning of the run when particles are nearly uniformly distributed, each process requires approximately the same amount of memory. Using the data from Table 2.3, we estimate the initial maximum memory usage to be $M_{\rm P}+M_{\rm HC}+M_{\rm G}+M_{\rm PM}+M_{\rm FFT} = (11+0.23+0.5+2+1)\times 800^3/n_{\rm pr}\times 4$ bytes $= 359.6$ MB, compared with the measured value of 366.5 MB. Most of the difference comes from the table of HC entries (3.4 MB) plus slight variations in the HC mesh and particle storage among processes.

At the end of the run the domains of each process have changed substantially. All processes require at least $M_{\rm G}+M_{\rm PM}+M_{\rm FFT} = (0.5+2+1)\times n^0 n^1 n^2 \times 4/n_{\rm pr} = 85.4$ MB, compared with the measured minimum value of 98.3 MB. The maximum amount of memory varies substantially and is hard to control at the final timesteps. We limit the maximum memory using the techniques mentioned in §2.6.4. During the last timestep 569, the maximum memory usage was reached on process 27. During this timestep process 27 changed the volume of its domain from 1.23×10^6 HC cells to 0.94×10^6 cells, which reduced the number of particles in its local region from 17.7×10^6 to 14.2×10^6. The number of PP boundary layer particles received by this process (see § 2.6.3) during the same timestep was 1.1×10^6. The measurement of 857.8 MB compares with the predicted value (before repartitioning) $M_{\rm P}+M_{\rm HC}+M_{\rm G}+M_{\rm PM}+M_{\rm FFT} =$

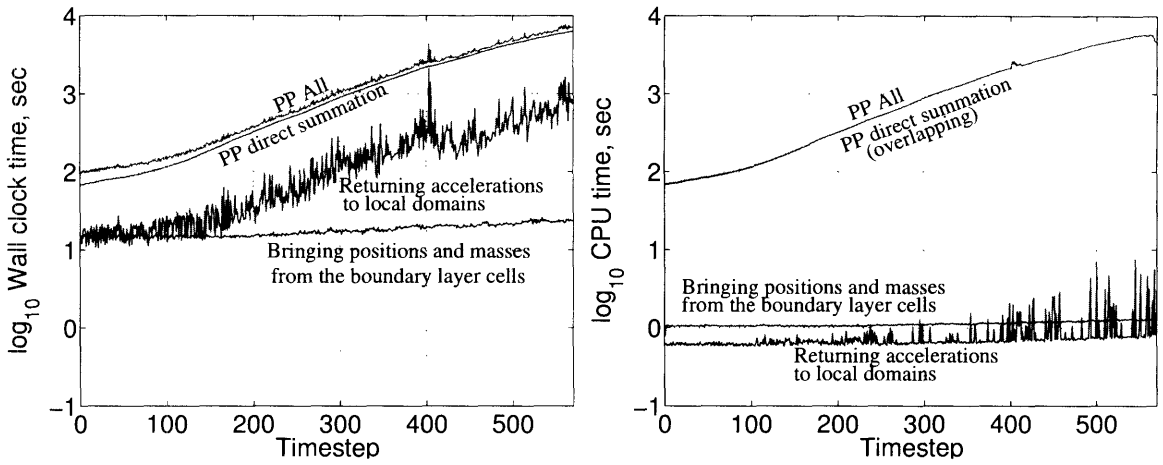

Figure 2-20: Structure of the wall clock time (left) and CPU time (right) of the PP force computation. The times are averaged over processes.

$(778.5 + 24.6 + 12.8 + 51.2 + 25.6) \times 10^6 = 851.3$ MB. Again the table of HC entries (3.4 MB) makes up most of the difference.

The reader will notice the similarity between Figures 2-18 and 2-21. The maximum memory usage tracks the volume of HC local regions because the most variable memory element is the number of particles, which correlates strongly with the number of HC cells. Strikingly, the CPU time for the PP pair summation does not correlate well with the number of particles or the maximum memory usage. At the last timestep, process 27 (which used 857.7 MB) took 6037 seconds of CPU time for the pair summation while process 48 (which used 98.1 MB) took 7152 seconds. This is a measure of the success of load balancing, which attempts to equalize CPU time rather than memory usage across all processes.

2.7.5 Parallel Scalability

A key test of any parallel code is its scalability as the problem size and/or number of CPUs increase. For a fixed problem size, if the wall clock time scales inversely with the number of CPUs, then one may use more CPUs to realize the proverb "many hands make light work." The ideal inverse scaling is readily achievable with so-called "embarrassingly parallel" codes that require little or no communication, but high efficiency is much more difficult to achieve for algorithms as complex as P^3M. Even if a code does not scale perfectly with a fixed problem size, it may scale well when the problem size is increased, enabling one to make effective use of supercomputers with hundreds or thousands of processes to perform very large simulations.

We tested the scalability of `llp3m-hc` using two problem sizes (288^3 and 384^3 ΛCDM in a 200 Mpc box with Plummer softening length $\epsilon = 0.1$ Mpc, evolved to redshift zero, taking 634 and 657 timesteps, respectively) and a range of numbers of computing nodes as shown in Table 2.4. Each computing node has two CPUs. The runs have either two processes per node (one per CPU, Runs 4a,b) or four processes

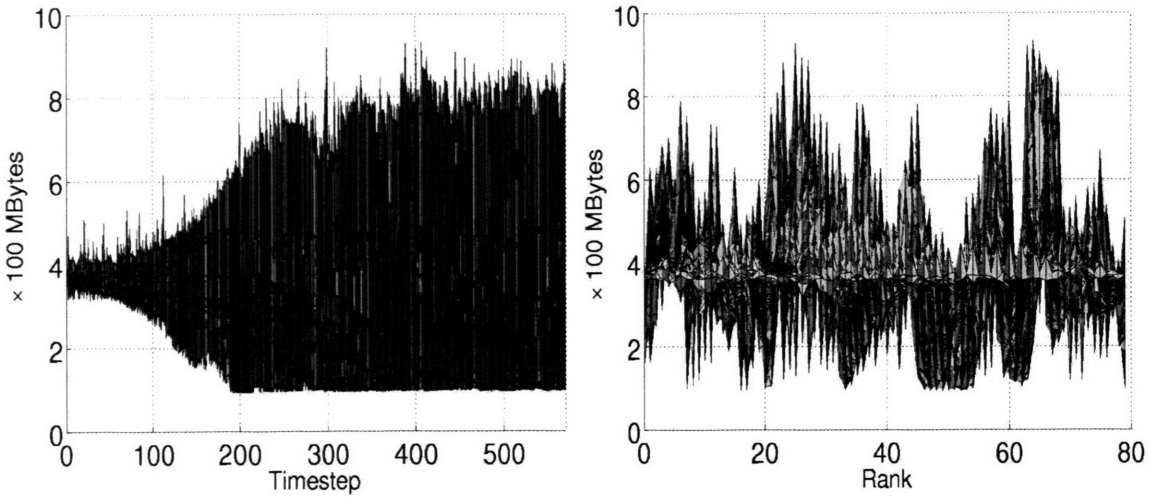

Figure 2-21: Maximum total memory allocated by any node, as a function of timestep (left panel) and process rank (right panel).

per node (two per CPU, using Intel hyperthreading). For perfect scalability, the times in the last column would be equal for simulations of the same grid size $N_{\rm gr}$.

From Table 2.4 we may draw several conclusions. First, `llp3m-hc` does not scale perfectly like an embarrassingly parallel application. On the other hand, increasing the number of processes up to 80 leads to a steadily decreasing wall clock time. Comparing Runs 3a and 3g, we see that for up to 48 processes, the wall clock time scales as $n_{\rm pr}^{-0.86}$. Hyperthreading also gives a significant speedup. Comparing Runs 3f and 4b, which have the same total number of processes but different numbers of compute nodes, we see that hyperthreading improves the code performance by a factor 1.62. We also see that the code scales reasonably well as the problem size is increased. Comparing Runs 3f and 5b, the wall clock time is proportional to $N^{1.74}$ where N is the number of particles. When the wall clock time is dominated by PP pair summation, we expect scaling as N^2.

The most significant deviations from perfect scalability arise with the largest numbers of processes, in particular Runs 3h, 3i, and 5d. These arise from load imbalance, as shown in Figure 2-22. A significant increase in load imbalance shows up after timestep 500 in Runs 3 and timestep 600 in Runs 4 due to the formation of a dense dark matter clump. When the number of processes is sufficiently large, this leads to one or a few HC cells beginning to take as much time for PP pairwise summation as the average time for the other processes. According to equation (2.31), the result is a growing residual load imbalance. Scalability breaks down beyond a certain number of processes, given by equation (2.32). Once the performance saturates, the instantaneous and residual load imbalance match because it is no longer possible to improve the load balancing by rearrangement of the partitioning.

Although the performance of `llp3m-hc` is limited by the PP pair summation and not by the PM force computation, it is worth recalling that, because the current code

Table 2.4: Scalability Runs.

Run	$N_{\rm gr}$	Nodes	Proc./node	$n_{\rm pr}$	(Wall Clock Time)×Nodes
3a	288^3	1	4	4	79.2h
3b	288^3	2	4	8	82.8h
3c	288^3	3	4	12	87.0h
3d	288^3	4	4	16	91.4h
3e	288^3	8	4	32	108.1h
3f	288^3	10	4	40	104.9h
3g	288^3	12	4	48	111.8h
3h	288^3	16	4	64	126.0h
3i	288^3	20	4	80	152.3h
4a	288^3	10	2	20	148.6h
4b	288^3	20	2	40	170.2h
5a	384^3	5	4	20	421.2h
5b	384^3	10	4	40	470.7h
5c	384^3	14	4	56	503.5h
5d	384^3	20	4	80	589.1h

uses blocking sends and receives to pass data between the particle and grid structures, the PM time also scales imperfectly. When we implement adaptive P^3M, the PP time will decrease significantly so that the PM time becomes a significant fraction of the total wall clock time. To improve the parallel scaling, it will be important to implement non-blocking communication for the PM particle/grid messages.

2.8 Conclusions

Parallelizing a gravitational N-body code involves considerably more work than simply computing different sections of an array on different processors. The extreme clustering that develops as a result of gravitational instability creates significant challenges. A successful parallelization strategy requires careful consideration of CPU load balancing, memory management, communication cost, and scalability.

The first decision that must be made in parallelizing any algorithm is how to divide up the problem to run on multiple processes. In the present context this means choosing a method of domain decomposition. Because P^3M is a hybrid algorithm combining elements of three-dimensional rectangular meshes and one-dimensional particle lists, we chose a hybrid method of domain decomposition. A regular mesh, distributed among the processes by a simple slab domain decomposition, is used to obtain the PM force from the mesh density. A one-dimensional structure — the Hilbert curve – is introduced to handle the distribution of particles across the processes and to load balance the work done on particles.

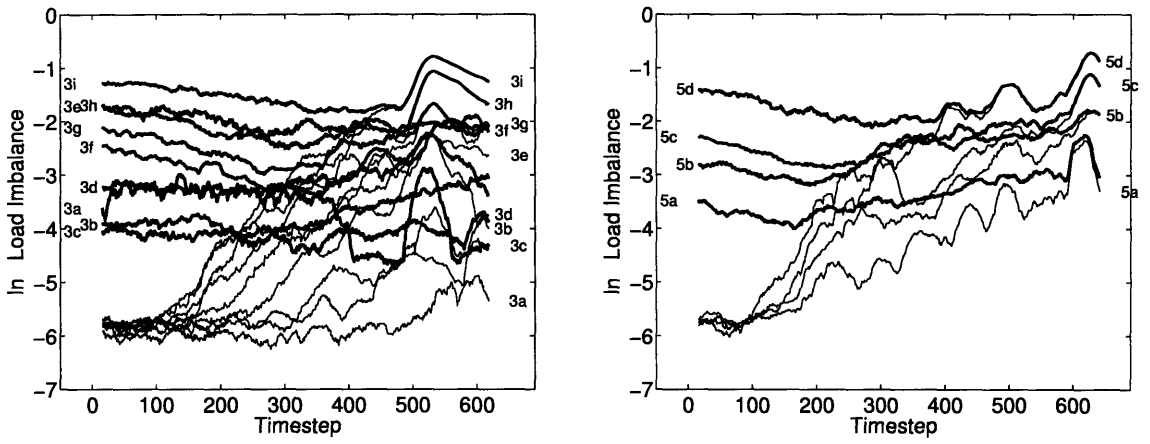

Figure 2-22: Instantaneous (heavy lines) and residual (thin lines) load imbalance as a function of timestep for Runs 3 ($N_{\rm gr} = 288^3$, left) and 5 ($N_{\rm gr} = 384^3$, right). The individual runs are labelled.

Implementing Hilbert curve domain decomposition in a particle code is the major innovation of our work. To take full advantage of it we had to employ a number of advanced techniques. First, in §2.4 we devised a discrete algorithm to find the nearly optimal partitioning of the Hilbert curve so as to achieve load balance, the desirable state in which all processors have the same amount of work to do. This is a much greater challenge in a hybrid code than in a purely mesh-based code such as a hydrodynamic solver or a gridless particle code such as the tree code. We then made the domain decomposition dynamic by repartitioning the Hilbert curve every timestep, allowing us to dynamically maintain approximate load balance even when the particle clustering became strong.

In §2.5.2 we presented a fast method for finding the position of a cell along the Hilbert curve given its three-dimensional location. This procedure allows us to access arbitrary cells in a general irregular domain by a lookup table much faster than using the special-purpose Hilbert curve function of [39].

In §2.6.1 we introduced run-length encoding to greatly reduce the communication cost for transferring information between the particle and mesh structures required during the PM force computation.

In §2.5.1 we optimized the particle distribution within each process so as to improve the cache performance critical for efficient pair summation in the PP force calculation.

By choosing the domain decomposition method appropriate for each data structure, and by implementing these additional innovations, we achieved good load balance and scalability even under extreme clustering. The techniques we introduced for effective parallelization should be applicable to a broad range of other computational problems in astrophysics including smooth-particle hydrodynamics and radiative transfer.

Tests of our algorithm in §2.7 showed that we achieved our goals of scalability and load balance, with two caveats mentioned at the end.

In Figure 2-13 we demonstrated the importance of using a dynamic three-dimensional domain decomposition method instead of a static one-dimensional slab decomposition. The latter method is unable to handle extreme spatial inhomogeneity.

Next, we performed a long 800^3 ΛCDM simulation (performed on only 20 dual-processor computing nodes) to thoroughly test the load balancing algorithm. The average load imbalance for this simulation run with 80 processes was only 12%, meaning that 12% of the total wall clock time of all the CPUs was wasted. While not perfect, this is very good performance for the P^{3}M algorithm. The largest cause of load imbalance over most of the simulation was our inability to predict the total CPU time of the next timestep on each process because of variations in cache memory usage.

Finally, we tested the limits of scalability by performing the set of runs in Table 2.4. For up to 48 processes the code performed with very good parallel speedup — the wall clock time scaled as $n_{\rm pr}^{-0.86}$ for $n_{\rm pr}$ processes, as compared with $n_{\rm pr}^{-1}$ for perfect scalability.

Our tests revealed two limitations to scalability that will be addressed in a later chapter presenting an adaptive P^{3}M algorithm. First, the current code uses blocking communication for sending data between the particle and grid structures in the PM force calculation. In other words, some processes sit idle waiting for others to complete their communications requests. This inefficiency, while small when PP forces are expensive to compute, will become more important when adaptive mesh refinement reduces the PP cost. The solution is to restructure the communication to work with non-blocking sends and receives.

Finally, we observed our code to become inefficient when a handful of Hilbert curve cells (out of millions in the entire simulation) begin to dominate the computation of PP forces. Because a non-adaptive code does not allow refinement of one cell, a single process must handle these extremely clustered cells even if the other processes have to wait idly while it finishes. The solution to this problem is simply to use adaptive refinement. In a later chapter we present an algorithm for scalable adaptive P^{3}M building upon the techniques introduced in the current chapter.

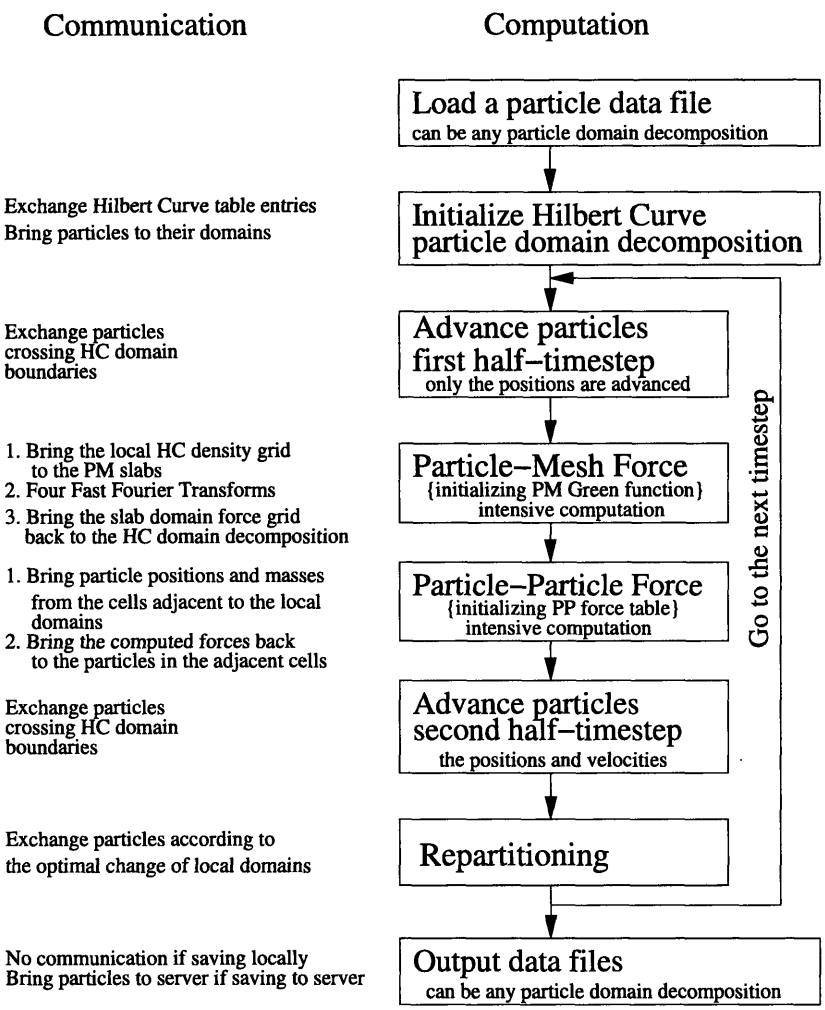

Figure 2-23: Block diagram of the parallel P^3M code `llp3m-hc`.

2.9 Appendix: Code Overview and Variables

Figure 2-23 presents a block diagram of our parallel Hilbert curve domain decomposition code `llp3m-hc`. The code may run on any number of processes n_{pr} (this is not restricted to being a power of 2). The code is written in ansi C with MPI calls. Excluding FFTW, it consists of about 33,000 lines of code. This Appendix gives an overview of the code guiding the reader to the relevant parts of the main text in Chapter 2.

The code begins by loading particle data from one or more files. At the beginning of a simulation, these files contain the initial conditions. A simulation may also be started using particle data that have already been evolved. The particle data may be either in one file on the cluster server or they may be in multiple files, one stored on each cluster compute node.

The next step is to initialize the Hilbert curve for domain decomposition based on the particle distribution, as described in §2.3.3. The `llp3m-hc` code stores particle

data (e.g. positions and other variables as described in §2.2.3) differently than mesh data (e.g. density). Mesh-based data are stored on a regular PM mesh which is divided by planes into a set of thick slabs, one for each parallel process. Particle data are organized into larger cells called Hilbert curve (HC) cells. (These cells have a size just slightly larger than the cutoff radius for the particle-particle or PP short-range force.) The cells are then connected like beads on a necklace by a closed one-dimensional curve called a Hilbert curve. The Hilbert curve initialization step computes and stores the information needed to determine the location of every bead on the necklace, that is, it associates a one-dimensional address with each HC cell.

Once the Hilbert curve is initialized, the Hilbert curve is cut into a series of segments, each segment (called a HC local region) containing a set of HC cells and their associated particles. Each parallel process owns one of the local regions. The particles are thus sent from the process on which they were initially loaded to the process where they belong. When restarting a run on the same nodes, the particles are already on the correct processes. When starting a new simulation, the partitions are set with equal spacing along the Hilbert Curve and the particles are sent to the appropriate processes.

This method of assigning particles to processes based on their position along a one-dimensional curve of discrete segments is called Hilbert curve domain decomposition and it is explained in §2.3. The organization of particles within a process is described in §2.5.

After these initialization steps the code integrates the equations of motion given in §2.2.2 using a leapfrog scheme presented in §2.2.4. First the positions are advanced one-half timestep, and if they cross HC local region boundaries they are moved to the correct process.

Next, gravitational forces are computed. Most of the work done by the code is spent computing forces. The interparticle forces are split into a long-range particle-mesh part computed on the mesh and interpolated to the particles, plus a short-range particle-particle correction, as described in §§2.2.5, 2.2.6, and 2.6. Most of the communication between processes occurs during these steps. If the particle-mesh Green's function has not yet been computed, it is computed just before the first PM calculation. The Green's function is essentially the discrete Fourier transform of r^{-2}, modified by an anti-aliasing filter to reduce anisotropy on scales of the PM mesh spacing. After the particle-mesh forces are computed, they are incremented by the particle-particle forces (the most time-consuming part of P^3M). After the forces are computed, velocities and then positions are advanced to the end of the timestep. Once more, particles that cross HC local region boundaries are transferred to the correct process.

After the particles have moved, the cuts along the Hilbert curve are moved so as to change the segment lengths and thereby change the domain decomposition. This step is called repartitioning. Its purpose is to ensure that, as much as possible, each process takes the same amount of time to perform its work as every other process, so that processes do not sit idle waiting for others to finish their work. (Certain operations, like the FFT, must be globally synchronized.) When this ideal situation is met, the code is said to be load balanced. Repartitioning is performed every

timestep to optimize load balance, as explained in §2.4.

At the end of the integration step, the code generally loops back to advance another step. Periodically the code also outputs the particle data, usually writing in parallel to local hard drives attached to each compute node.

Table 2.5 presents a list of frequently used symbols and variables in the code.

2.10 Moore's Hilbert Curve Implementation Functions

Working with a Hilbert (space-filling) curve requires a mapping from HC index h to HC cell position c and vice versa. [39] implemented C functions that accomplish these mappings.

The most straightforward implementation of the Hilbert curve is too slow, since a Hilbert curve is defined recursively by its self similarity. Moore's implementation is based on a much faster non-recursive algorithm of [12].

A one-to-one correspondence between a cell and the HC index is given by the following functions of Moore's implementation:

$$\begin{aligned} h &= \texttt{hilbert\_c2i}(d, m, \mathbf{c}) \equiv \mathcal{H}_d(\mathbf{c}) \\ \mathbf{c} &= \texttt{hilbert\_i2c}(d, m, h) \equiv \mathcal{H}_d^{-1}(h) \ . \end{aligned} \tag{2.44}$$

The Hilbert curve index h is of type `long long unsigned` and $\mathbf{c}$ a vector of three integer indices giving the spatial coordinates of the cell. These two functions are inverse to each other. They are implemented for any spatial dimension d. For example for $d = 2$ in Figure 2-4, a function $\mathcal{H}_2^{-1}(0)$ will return the position of the curve's starting point, and the function $\mathcal{H}_2^{-1}(1)$ returns the position of the next cell along the curve. We verified that the resulting curve indeed provides a one-to-one mapping between the cell and its HC index preserving space locality for all HC mesh sizes up to 2048^3.

Table 2.6 shows the average measured CPU time to make one call to the HC function `hilbert_c2i` on a 2.4 GHz Intel Xeon processor. The time shown is compared with the average times to make other simple arithmetic operations or memory references. It is surprising how fast the implementation is: it takes just two minutes to make 512^3 Hilbert curve function calls on a single processor. However, in comparison with a simple arithmetic operation or triple array dereferencing, it is very slow: An average `hilbert_c2i` function call is about 120 times slower than a triple array dereferencing for the 512^3 HC mesh; the function call time increases linearly with the increase of the Hilbert curve order m as $(4.10 + 0.775m) \times 10^{-7}$ sec. We should therefore avoid using the HC implementation function calls when it is possible to use memory dereferencing instead. As we discuss in §§2.5.1 and 2.6.1 we successfully avoid multiple calls to `hilbert_c2i` during the force calculation and the particle advancement by proper organization of memory usage.

Notation	Code variables	Description
Serial and parallel codes		
L^0, L^1, L^2		The simulation box size in comoving Mpc, $L^i \equiv n^i \Delta x$.
N		The total number of particles in the simulation volume
Δx	`dx`	The PM mesh spacing, same in all dimensions
$\tilde{\epsilon}$	`epsilon`	Plummer softening length, in units of Δx
η_t	`etat`	Time integration parameter, usually $\eta_t = 0.05$
$\tilde{R}_{\max}$	`cr.max`	PP-force length, in units of Δx, typically 2.78
n^0, n^1, n^2	`n0..n2`	The size of the simulation box, in units of PM cells
n_c^0, n_c^1, n_c^2	`ncm0..ncm2`	The size of the simulation box in chaining mesh cells
$N_{\rm gr}$	`ngrid`	$= n^0 n^1 n^2$, the total number of PM grid points
$\Delta \tilde{x}_c^i$	`cr.len0..cr.len2`	Chaining-mesh grid spacing along three dimensions
	`pa`, `pa_f`	Starting and finishing pointers of particle array [`pa`, `pa_f`)
	`pa_fa`	Pointer to the end of the preallocated particle array, equals `pa_f` in the serial code. In the parallel code `pa_fa` $\geq$ `pa_f`.
Parallel codes only		
	`sloc[i]`	Starting index of FFTW slab i for the FFT plan
	`nloc[i]`	Thickness of FFTW slab i. The whole slab on process i has size $n^0 n^1$`nloc`$[i]$, where $i = 0 \ldots n_{\rm pr} - 1$
n_h^0, n_h^1, n_h^2	`hc_n0`	The size of the simulation box in Hilbert curve (HC) mesh
	`...hc_n2`	cells, per dimension
$N_{\rm HC}$		$= n_h^0 n_h^1 n_h^2$, the total number of HC mesh cells.
$N(i)$		The number of particles local to process i
$n_{\rm pr}$	`wk.nproc`	Number of worker processes, those containing particle data
$\mathbf{c}$		Coordinates of a cell in the Hilbert curve mesh
h		A Hilbert curve index
$\tilde{r}$		A raw Hilbert curve index
$\mathcal{H}(\mathbf{c})$	`hilbert_c2i`	Mapping between the HC index h and the cell's coordinates
$\mathcal{H}^{-1}(h)$	`hilbert_i2c`	The inverse of the above mapping
$\mathcal{H}_{\rm r}(\mathbf{c})$		Mapping between the HC raw index h and the cell's coordinates
m		HC order: the number of cells in the HC mesh is 2^{md}, $d = 3$
$\Delta \tilde{x}_h^i$		HC mesh spacing along dimension i, in units of Δx
L_h^i		HC local region of process $i \in [0, n_{\rm pr})$
	`hc_stg`	3-d ragged array with gaps of the cells of the L_h^i
K		The number of entries of the HC into the simulation volume
$h_{\rm eb}^k, h_{\rm en}^k$		The HC index of the k-th entry of the curve into the simulation volume $k \in [0, K)$, and the number of the HC cells that follow contiguously inside the simulation volume along the curve
$h_{\rm b}(i), h_{\rm n}(i)$		The HC index of the bottom partition and number of cells on process i
$r_{\rm b}(i), r_{\rm n}(i)$		Same, with the raw index

Table 2.5: Frequently used variables.

Operation within triple `for` loop	CPU time per call, 10^{-9} sec
nothing (bare triple `for` loop)	7.75
inline multiplication (innermost integer index squared)	12.8
arithmetic function call (innermost integer index squared)	18.57
triple array dereferencing	16.29
`hilbert_c2i` function call ($m = 9$)	1056.
`hilbert_i2c` function call ($m = 9$)	920.

Table 2.6: CPU time averaged for 512^3 `hilbert_c2i` function calls to ($m = 9$ bits per dimension)

Chapter 3

Scalable and Load Balanced Adaptive P^3M Cosmological N-body Code

3.1 Introduction

When the P^3M technique for gravitational force calculation in an N-body system was introduced [21], it became clear that the *particle–particle* (*PP*) (short range) force calculation is a major problem for N-body simulations at the late stages of the evolution of an N-body system. The PP-force is defined as a force correction performed over the pairs of particles separated by the distance less than the *PP-cutoff distance* in order to achieve the desired interparticle force law.

Calculation of PP-force can be achieved by direct summation, in which case the number of terms in the direct summation equals the number of pairs of particles in the whole simulation volume whose separation is smaller than the cutoff distance. As the particles cluster due to their mutual gravity, the total number of direct summations needed each timestep for PP-force calculation grows steeply. Indeed, since more and more particles become members of one of the clusters in the simulation box and the particle number density in the clusters steeply grows with time, given a random particle within the simulation volume the expectation number of the other particles within the PP-cutoff distance steeply grows as well.

In Chapter 2, we introduced a scalable and load balanced parallel code (`l1p3m-hc`), where the PP-forces are computed by direct summation. The P^3M test runs presented there showed very good load balance achieved for the runs all the way through the regime of extreme clustering. However, the problem of steeply growing total workload of computing the PP-forces can not be solved by pure load balancing and scalability. Indeed, despite a very low degree of load imbalance of 12% in the test run for ΛCDM universe with 800^3 particles, the heavy total load of computing the PP-forces effectively stopped the simulation progress at high clustering. The PP direct summations take more than 98% of the whole simulation time at redshift $z = 0.45$, at which point the PP-force computation time per timestep reached two hours as computed by our

cluster.

In this Chapter we present the extension `llap3m-hc` of scalable and load balanced parallel P^{3}M algorithm described in Chapter 2 to include an *adaptive* method for particle–particle force calculation aimed at reducing the total workload at high clustering. Adaptive particle–particle forces and the method for their computation were introduced by Couchman [13], however our technique has many important differences from his.

3.2 Adaptive PP-force

In order to describe exactly how the adaptive-PP forces are introduced, we make a few definitions in the following.

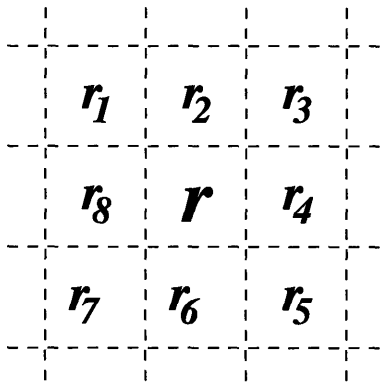

Figure 3-1: Two dimensional illustration of chaining mesh assignment for short range force calculation for cell i. By definition, $B^{\mathrm{PP}}(r) = \{r, r_1, \ldots, r_4\}$ and $B^{\mathrm{APP}}(r) = \{r, r_1, \ldots, r_8\}$.

Each time before PP-force computation is performed (either adaptively or by direct summation) all the particles within the simulation volume are sorted into the *chaining mesh* cells, whose spacing either equals or slightly exceeds the PP-force cutoff distance. Chaining mesh is used in order to select all the pairs of particles within the simulation volume whose separation is within the PP-cutoff distance. Given a random chaining mesh cell r (see Figure 3-1 for two dimensional illustration), the particle data within the cell r and all the adjacent cells $r_1, \ldots, r_8$ is necessary and sufficient for complete computation of the PP-forces acting on all the particles within the cell r. We define the set $B^0(r) \equiv \{r_1, \ldots, r_8\}$.

While computing PP-force by direct summation for cell r, we avoid the double counting effect in pair summations by using only half of the adjacent cells $B^1(r) \equiv \{r_1, \ldots, r_4\}$ for the PP-force computation. By this procedure, the PP-forces are completely computed for all the particles within r only after the same procedure is finished for the cells r_5, r_6, r_7, r_8, since for example $r \in B^1(r_5)$. We define the *PP*

direct summation simulation volume by

$$B^{\mathrm{PP}} \equiv r \cup B^1(r) \tag{3.1}$$

By introducing the chaining mesh we are able to formally split the total short range force computation for all particles within the simulation volume into additive contributions due to each of the cells within the chaining mesh. In Chapter 2 we defined the Hilbert curve (HC) mesh used for particle domain decomposition for the parallel code to be identical to the chaining mesh, therefore we will not distinguish between them in the further text. The *effective cell workload* w is defined as the expected amount of the wall clock time required by force computation associated with a given cell within the next timestep. The workload of a cell is approximated using the direct timing measurements performed during the latest timesteps (see the definition of the effective cell workload in Section 2.4.1 of Chapter 2 for more details). The sum of the cell workloads for all the HC-cells local to a process yields the *process effective workload*, whose sum over all processes yields the total simulation effective workload for a given timestep.

The PP-force is computed adaptively for the chaining mesh cells whose PP-force workloads w^{PP} are extraordinarily high. The adaptive method works by transforming the cost of the PP-direct summation w^{PP} for the given chaining mesh cell into a combination of *fine PM* (*FPM*) and *fine PP* (*FPP*)-force calculations computed within the *fine mesh simulation volume* $r \cup B^0(r)$ corresponding to the chaining mesh cell r for which adaptive short range force computation takes place. The adaptive PP-force computation is expected to be advantageous over the computation by pure direct summation when

$$w^{\mathrm{APP}} \equiv w^{\mathrm{FPP}} + w^{\mathrm{FPM}} < w^{\mathrm{PP}} \tag{3.2}$$

and when the splitting into the two components does not result in a significant loss in the accuracy of the computed PP-force. The procedure for computing the PP-forces adaptively is generically referred to as *mesh refinement*.

The splitting of the PP-force into FPM and FPP components is in many ways analogous to the way the interparticle force is split into the PM and PP-components in the original $\mathrm{P^3M}$. Just as for the PM-forces, the Fast Fourier Transform technique is used for FPM-force computation. Just as for the PP forces, the direct summation is used for FPP force computation. In order to compute the FPM- and FPP- forces, in Section 3.2.1 we define the *fine density mesh* and the *fine chaining mesh*, in analogy to the density and chaining meshes used for PM and PP. The latter will be referred to in the further text as *coarse density mesh* and *coarse chaining mesh* in order to distinguish them from the fine density and chaining meshes.

In contrast to the forces computed by direct summation, the cost of computing the FPM-forces is dominated by the Fast Fourier Transform. This cost is independent of the number of particles within the fine mesh simulation volume. The advantage of mesh refinement for coarse chaining mesh cells with heavy workloads is reached because the workload of PP-force computation scales roughly as a square of the number of particles within the PP-force simulation volume, while the combination of

FPM- and FPP- forces may result in lower workload because the FPM-workload is roughly constant, while the FPP-workload, although performed by direct summation, uses a much finer chaining mesh, leading to the lower workload.

As we will see in Section 3.4, the timing advantage $w^{\rm PP}/(w^{\rm FPP}+w^{\rm FPM})$ of computing the PP-forces adaptively reaches orders of magnitude for heavy workload cells. Using a good approximation $w^{\rm FPP} \approx w^{\rm FPM}$ for those cells (see Section 3.3.3) this leads to $w^{\rm FPM} \ll w^{\rm PP}$. Now, as we mentioned above, while processing a cell r for PP-force computation by direct summation, we only use half of the surrounding cells $r \cup B^{\rm PP}(r)$ to avoid the double counting. One should expect that if we were to use all the adjacent cells $B^0(r)$ for direct summation, the expected PP-direct summation workload $w^{\rm PP}$ for this cell would likely double while reducing the direct summation cost of the other cells in the simulation volume by the same amount because the cell r will then be excluded from their direct summation to avoid double counting. On the other hand, if we are computing the PP-forces adaptively for cell r, adding all the adjacent cells changes the workload only by a small amount $w^{\rm FPM} \ll w^{\rm PP}$, while reducing the total workload of the other cells by the amount of the order of $w^{\rm PP}$. In the current implementation `l1ap3m-hc` we adopt a greedy strategy: whenever we perform adaptive PP-force computation for cell r, we compute the short range components for all the adjacent cells with the expectation that this will reduce the total workload of the problem.

The fine mesh simulation volume for adaptive-PP force calculation for cell r is therefore defined as

$$B^{\rm APP}(r) \equiv r \cup B^0(r) \tag{3.3}$$

Upon the completion of the adaptive-PP force computation for cell r, the PP-forces on all the cells contained within r are computed completely (which is different from PP by direct summations).

3.2.1 Mesh Refinement Geometry

Here we define the geometry of the mesh refinement and introduce the mesh refinement free parameters. We use tilde for dimensional quantities expressed in code units, and use the same notations as in Chapter 2 unless otherwise noted.

As we discussed in Chapter 2 the spacing of coarse PM-density mesh is set to $\Delta\tilde{x} = 1$ in code units. The whole simulation box has side lengths $L^i = n^i \Delta\tilde{x} = n^i$ ($i = \{0,1,2\}$), where n^i is the PM-density mesh size along the i-th direction. The total PM-density mesh size is given by $N_{\rm gr} \equiv n^0 n^1 n^2$. The spacing $\Delta\tilde{x}^i_{\rm c}$ of the coarse chaining mesh is set to the minimum possible value greater than $\tilde{R}_{\rm c\,max}$, where $\tilde{R}_{\rm c\,max}$ is the short range cutoff distance for PP-forces (we added a subscript to the notation in Chapter 2 where we used instead $\tilde{R}_{\rm max}$ in order to make a distinction from the FPM-force cutoff distance soon to be introduced). The value of $\tilde{R}_{\rm c\,max}$, given by the required force resolution (see Section 3.2.4), is used to define the size of the coarse mesh along each of the directions $n^i_{\rm c} = [L^i/\tilde{R}_{\rm c\,max}]$, where the square brackets signify taking the integer part. The spacing of the coarse mesh cells is obtained by $\Delta\tilde{x}^i_{\rm c} = L^i/n^i_{\rm c} = L^i/[L^i/\tilde{R}_{\rm c\,max}]$. That is, we require the whole simulation box to

have side length L^i that are integer multiples of $\Delta\tilde{x}_c^i$.

Let us set up the mesh refinement parameters now. By definition, the cell r fine mesh simulation volume $B^{\mathrm{APP}}(r)$ has the side length of $\tilde{L}_f^i = 3\Delta\tilde{x}_c^i$. Since FFT is used for FPM-force calculation on the fine density mesh, its spacing $\Delta\tilde{x}_f^i$ must be the same in all directions in order to get the spherically symmetric FPM-forces.

Let us set a new constraint on coarse mesh cell spacing for simplicity of switching between different mesh refinement numbers (to be introduced further in this Chapter)

$$\Delta\tilde{x}_c^0 = \Delta\tilde{x}_c^1 = \Delta\tilde{x}_c^2 \equiv \Delta\tilde{x}_c \ . \tag{3.4}$$

This results in

$$\tilde{L}_f = \text{constant}, \quad n_f = \text{constant}, \text{ and } \Delta\tilde{x}_f = \tilde{L}_f/n_f \ , \tag{3.5}$$

where n_f is the size of the fine density mesh, being a free parameter for fine mesh refinement. The constraint (3.4) does not apply for the non-adaptive code `l1p3m-hc`.

The fine chaining mesh is defined in a completely analogous way to the similar procedure for the coarse chaining mesh. Given the maximum particle pairwise separation $\tilde{R}_{f\,max}$ at which the FPP-force is applied, the fine chaining mesh size can be set by

$$n_{fc} = [\tilde{L}_f/\tilde{R}_{f\,max}] \ . \tag{3.6}$$

However, as we will see in Section 3.3.4, this setting is not optimal since it does not in general satisfy a constraint given by equation (3.49) that is required in order to achieve certain particle data layout optimization. Equation (3.6) is modified in order to fit this constraint as

$$n_{fc} = \left[\tilde{L}_f/\tilde{R}_{f\,max}\right] - \mod\left([\tilde{L}_f/\tilde{R}_{f\,max}], 3\right) \ . \tag{3.7}$$

The cell spacing is then given by $\Delta\tilde{x}_{fc} = \tilde{L}_f/n_{fc}$, where n_{fc} is a multiple of 3.

We have thus far completely specified the mesh refinement geometry given two free parameters n_f and $\tilde{R}_{f\,max}$. The choice for their values is given both by the desired force accuracy and by timing performance (as will be shown in Sections 3.2.4 and 3.3). Similar to P^3M, one expects that the optimal choice for $\tilde{R}_{f\,max}$ is a roughly constant factor of the fine density mesh grid spacings which depends on n_f explicitly through equation (3.5). This explicit dependency is scaled out by introducing a new free parameter C_f as a substitute

$$C_f \equiv \frac{n_f \tilde{R}_{f\,max}}{\tilde{R}_{c\,max}} \ . \tag{3.8}$$

We are left with two free parameters for mesh refinement: n_f and C_f, required to completely specify fine density and chaining meshes given coarse chaining mesh spacing $\Delta\tilde{x}_c$.

3.2.2 Isolated Boundary Conditions with FFT-based Force

The computation of FPM-forces is based on Fast Fourier Transform on a periodic grid and therefore the computed forces have intrinsically periodic boundary conditions on a fine mesh simulation volume. The sum of FPM and FPP forces is intended to reproduce the PP-forces which have isolated boundary conditions. Here we introduce a method allowing us to take all the advantage of FFT for computing FPM forces while avoiding the effect of periodic boundary conditions in the computed forces.

Let us define for convenience the *force law* $\boldsymbol{\Theta}(\mathbf{r})$ as a function of force $\mathbf{F}$ acting between the particles of masses m_1 and m_2 within a pair by $\boldsymbol{\Theta}(\mathbf{r}) \equiv \mathbf{F}(\mathbf{r})/(Gm_1m_2)$, where $\mathbf{r}$ is the pairwise separation and G is the gravitational constant.

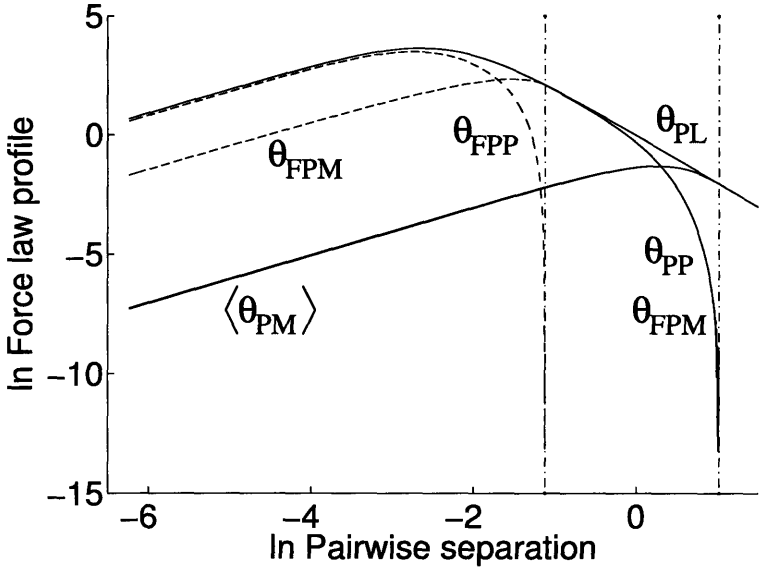

Figure 3-2: Force laws in a P^{3}M and adaptive P^{3}M simulation for parameters $\tilde{\eta} = 3.3$ (defined in Section 3.2.3), $\tilde{R}_{\mathrm{c\,max}} = 2.78$, $n_{\mathrm{f}} = 128$, $C_{\mathrm{f}} = 15$ and Plummer $\tilde{\epsilon} = 0.1$. This Figure is schematically applicable for any reasonable choice of parameters. The *dashed lines* show the splitting of PP-force into FPM- and FPP-components (Θ_{FPM} and Θ_{FPP}). The Plummer force law is marked with Θ_{PL}. Also shown is the average PM force law $\langle\Theta_{\mathrm{PM}}\rangle$ described in Section 3.2.5. Neglecting the error term, the Plummer force law is split into the sum of $\langle\Theta_{\mathrm{PM}}\rangle$ and Θ_{PP}.

Quantitatively, the force refinement is performed by splitting (see Figure 3-2) of the PP-force

$$\boldsymbol{\Theta}_{\mathrm{PP}}(\tilde{\mathbf{r}}) = \boldsymbol{\Theta}_{\mathrm{FPM}}(\tilde{\mathbf{r}}) + \boldsymbol{\Theta}_{\mathrm{FPP}}(\tilde{\mathbf{r}}) + \Delta\boldsymbol{\Theta}_{\mathrm{err}}(\tilde{\mathbf{r}}), \quad |\tilde{\mathbf{r}}| < \tilde{R}_{\mathrm{c\,max}}\,, \tag{3.9}$$

where $\Delta\boldsymbol{\Theta}_{\mathrm{err}}$ is the force error.

Note that this is analogous to the originally introduced P^{3}M splitting

$$\boldsymbol{\Theta}_{\mathrm{P3M}}(\tilde{\mathbf{r}}) = \boldsymbol{\Theta}_{\mathrm{PM}}(\tilde{\mathbf{r}}) + \boldsymbol{\Theta}_{\mathrm{PP}}(\tilde{\mathbf{r}}) \tag{3.10}$$

of the P^{3}M force into a sum of PM and PP forces. The P^{3}M force is designed to approximate the Plummer law $\boldsymbol{\Theta}_{\rm PL}(\tilde{\mathbf{r}})$. The Plummer force law, given by

$$\boldsymbol{\Theta}_{\rm PL}(\tilde{\mathbf{r}}) \equiv \frac{\boldsymbol{\Theta}_{\rm PL}(\tilde{r})\tilde{\mathbf{r}}}{\tilde{r}}, \text{where } \Theta_{\rm PL}(\tilde{r}) \equiv -\frac{\tilde{r}}{(\tilde{r}^2+\epsilon^2)^{3/2}} \tag{3.11}$$

constitutes the *required force law* for P^{3}M forces.

The overall force splitting is shown in Figure 3-2 for a specific set of parameters.

The FPM-force is computed within the whole fine mesh simulation box by a convolution of particle density with the PP-force law $\Theta_{\rm PP}$ (see Section 3.2.3 for full details) which is zero beyond the PP-force cutoff separation $\tilde{R}_{\rm c\,max}$. Due to the convolution theorem, we have for the computed $\Theta_{\rm FPM}$

$$\boldsymbol{\Theta}_{\rm FPM}(\tilde{\mathbf{r}}) = 0 \quad \text{for} \quad |\tilde{\mathbf{r}}| > \tilde{R}_{\rm c\,max}\,. \tag{3.12}$$

This property allows us to completely avoid the effect of periodic boundary conditions for the computed FPM-forces by computing the FPM force in sequence of two independent steps, as explained below.

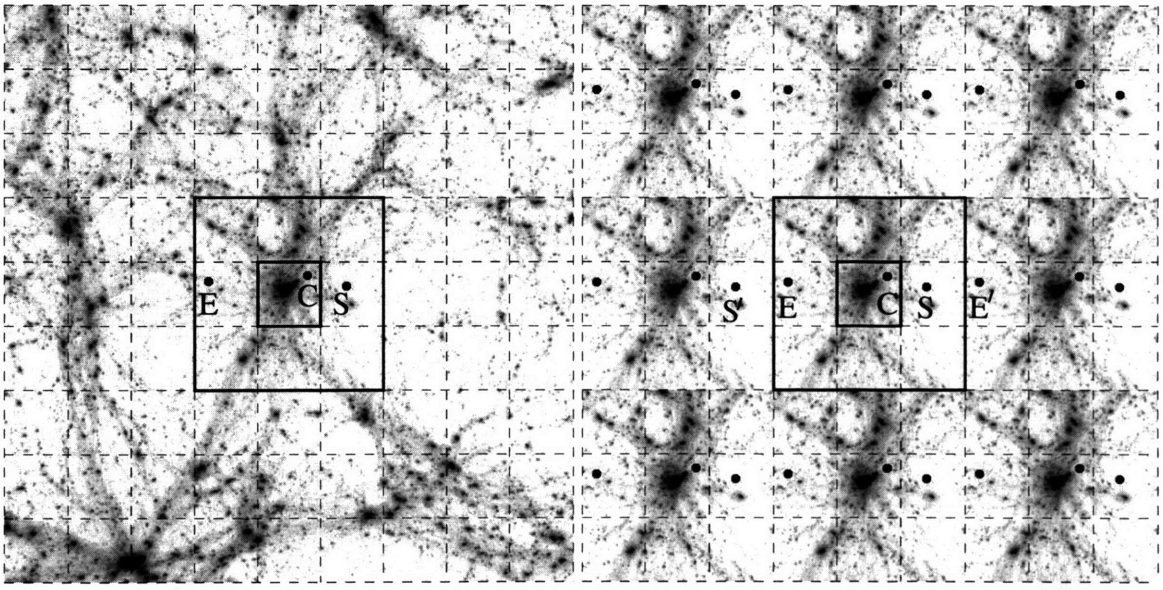

Figure 3-3: A portion of the simulation volume containing the mesh refined cell r (inner solid square). The left image shows the actual particle distribution of the simulation, while the right volume periodically replicates $B^{\rm APP}(r)$ showing what the FFT sees. Coarse chaining mesh cells are marked by the dashed lines. The mesh refined cell is within the inner solid square. The fine mesh simulation volume $B^{\rm APP}(r)$ is contained within the outer solid line rectangle. The particle C is within the mesh refined cell r. The particles E and S are outside the cell r but are still within the fine mesh simulation volume $B^{\rm APP}(r)$ for cell r. Of the particles E, C and S only C and S are positioned within the PP-cutoff separation range to each other $\overline{CS} \geq \tilde{R}_{\rm c\,max}$.

Let us clearly state the problem of periodic boundary conditions. In Figure 3-3 we

present as an example particle distribution within a small portion of simulation whole volume, occupied by 9×9 coarse mesh cells. Let us assume that the mesh refinement is performed for the cell r in the center. As we discussed at the end of Section 3.2, in the result of the adaptive-PP force computation *a)* the computed short range forces on a particle inside the cell r should correctly represent the contributions from all the other particles inside the mesh refinement simulation volume $B^{\rm APP}(r)$ and within the short range cutoff distance $\tilde{R}_{\rm c\,max}$; and *b)* the particles inside the adjacent cells $B^0(r)$ should get the PP-force update due to all the particles inside cell r that are within the maximum cutoff separation. According to the above points, of the three particles E, C and S considered and shown in Figure, the particle C gets a force contribution only due to particle S. The Particle S in turn gets a force contribution only due to particle C. The particle E does not get a force contribution from either of these particles.

Now, if we use all the particles within $B^{\rm APP}(r)$ for density interpolation for computing FPM forces, then, due to the periodic boundary conditions intrinsic to FPM the resulting FPM force field will be periodic, as shown in the right panel of Figure 3-3. Due to the periodic extension of the force field, particle S gets a forbidden force contribution from the periodic extension E' of particle E since the distance $\overline{SE'}$ falls within the maximum PP-cutoff separation $\tilde{R}_{\rm c\,max}$. Particle E in turn gets a forbidden force contribution from the periodic extension S' of particle S.

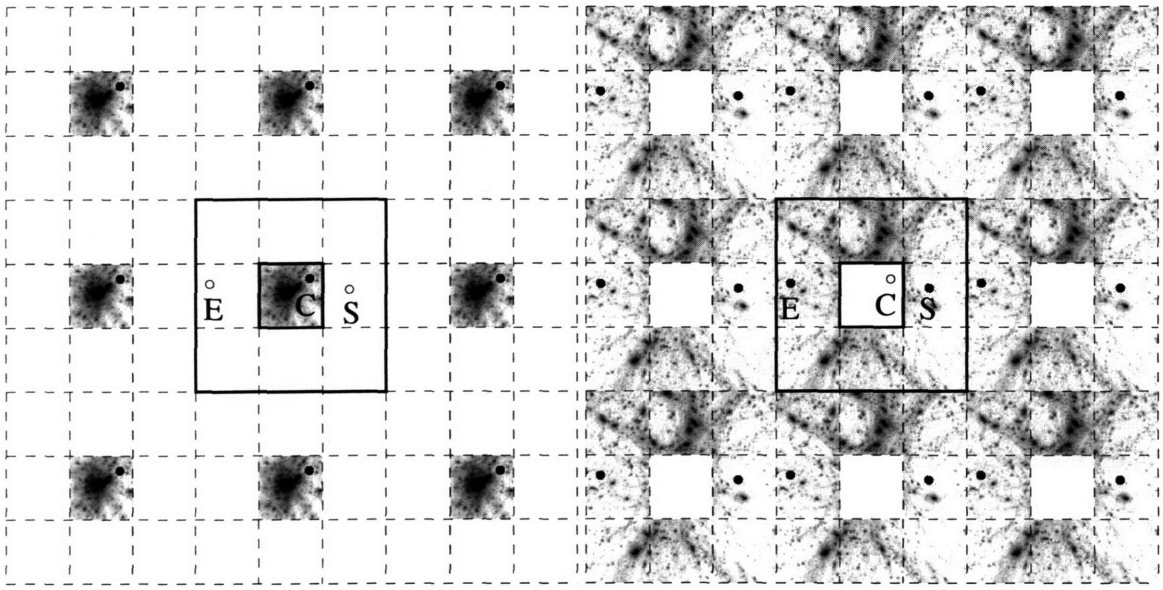

Figure 3-4: Density interpolation for the two steps of FPM calculation (*left* and *right* panels).

The deleterious effect of periodic boundary conditions for FPM is avoided by performing the density interpolation in the sequence of the following two steps.

First, we use only the particles within the central coarse mesh cell as shown on the left panel in Figure 3-4 to get the density interpolation on the complete fine density mesh. The resulting force field is then used to increment the forces for all the particles

within $B^{\rm APP}(r)$. Even though the force field is periodic, as shown in the figure, the effect of the periodically extended regions of non-zero density field on computed forces for particles within the fine mesh simulation volume is zero since those regions are all more distant than the cutoff distance $\tilde{R}_{\rm c\,max}$ from any particle within fine mesh simulation volume. During this step, the particle S gets a correct short range force contribution from the particle C.

During the second step (see right panel of Figure 3-4) we use all the particles within the fine mesh simulation volume excluding the central cell for density interpolation. The resulting FPM-forces are then used exclusively for incrementing the forces on the particles in the central cell. Since the periodically extended regions of non-zero density are all further away from the cell r than $\tilde{R}_{\rm c\,max}$, their force contribution to the particles within the central cell is zero. During this step, the particle C gets a PP-force contribution from the particle S.

The combination of these two steps leads to an accurate representation of the PP-force with the FPM-force for separations $\tilde{r} < \tilde{R}_{\rm c\,max}$. We have so far discussed only the FPM force since it is the only force in the adaptive-PP computed using FFTs with periodic boundary conditions. In fact, the FPM force alone does not accurately represent PP-forces. At $\tilde{r} \leq \tilde{R}_{\rm f\,max}$, the computed FPM-force is systematically different from the PP-force law, and a short range correction is applied to get the correct PP-force law in the total. This correction is called the *FPP-force* and is performed by doing direct summation over the fine chaining mesh cells in analogy with the PP-force where the direct summation is performed over the coarse mesh cells. The FPP-force does not suffer from the effect of periodic boundary conditions.

We conclude in this Section that using the above procedure to compute the FPM-forces leads to the computed adaptive-PP forces that do not suffer errors from the periodic boundary conditions of the FFT.

3.2.3 Optimal Green's Function for a General Compact Field Potential

Whenever we use a PM-force algorithm for force computation, we define the *required force law*, or the exact force law that the algorithm is trying to approximate. The forces computed in the simulation suffer from grid effects and in practice show a different force law, which may have both random and systematic deviations from the required force law.

The main difference between FPM and PM is their required force law. The *PM required force law*, denoted by $\Theta^0_{\rm PM}(r)$, whose Fourier Transform is given by Eq. (3.17), is long ranged and is uniquely parametrized by the halo extent parameter η. The optimal Green function for computing the PM-forces is derived in [23]. In contrast to PM-forces, the *FPM required force law* is localized to short ranges $\tilde{r} \leq \tilde{R}_{\rm c\,max}$ has more free parameters. We will follow the techniques developed in [23] in order to derive the optimal Green function for the FPM-force. In this section we borrow their notations and Fourier transform convention.

We refer the reader to [23] for the full discussion on PM-force algorithm. Let us

however review some of their results below in order to simplify our derivation. Given the optimal Green function $G(\mathbf{k}_0)$ defined in the vector space of the main Brillouin zone and given the particle positions and masses, the Fourier transform of PM-force, after all the grid effects are taken into account, is given by

$$\mathbf{F}_{\mathrm{gg}}(\mathbf{k}) = 2M\mathbf{D}(\mathbf{k}_0)G(\mathbf{k}_0)\widehat{\rho}_{\mathrm{gs}}(\mathbf{k}_0)\widehat{W}(\mathbf{k}) \,, \tag{3.13}$$

where M is the size of the simulation box in grid spacings, $\mathbf{k}$ is the wave vector with all integer components, $\mathbf{k}_0$ is the reduced vector whose components are defined within the main Brillouin zone as $k_0^i = \mod(k^i, M)$, $\mathbf{D}(\mathbf{k}) = i\mathbf{k}$ is the gradient operator, $\widehat{\rho}_{\mathrm{gs}}$ is the Fourier component of density in the main Brillouin zone, which is expressed in terms of the true (continuous) Fourier transform on the complete discrete infinite vector space $\mathbf{k}$ as

$$\widehat{\rho}_{\mathrm{gs}}(\mathbf{k}_0) \equiv \sum_{\mathbf{b}} \widehat{\rho}_s(\mathbf{k}_0 + M\mathbf{b}) \,,$$

where $\mathbf{b}$ is a vector whose components (b^0, b^1, b^2) take all integer values. The sum over the Brillouin zones $\mathbf{b}$ is called aliasing. The continuous density ρ_{s}, obtained from particle positions and masses by smoothing with window-function $W(\mathbf{x})$ is defined as

$$\rho_s(\mathbf{x}) = \int d^3x W(\mathbf{x} - \mathbf{x}')\rho(\mathbf{x}') = \sum_{i=0}^{N-1} m_i W(\mathbf{x} - \mathbf{x_i}) \,. \tag{3.14}$$

The optimal Green function $G(\mathbf{k}_0)$ is found by the minimization of the mean squared force error produced at $\mathbf{x}$ due to a particle at $\mathbf{x}_1$, averaged over both the positions of source and test particles

$$\mathcal{F} = \int d^3x_1 \int d^3x |\mathbf{F}_{\mathrm{gg}}(\mathbf{x}) - \mathbf{F}(\mathbf{x})|^2 = M^{-3} \sum_{\mathbf{k}} \int d^3x_1 |\widehat{\mathbf{F}}_{\mathrm{gg}}(\mathbf{k}) - \widehat{\mathbf{F}}(\mathbf{k})|^2 \,, \tag{3.15}$$

which can be split into sums over the reduced wave vectors and the Brillouin zones

$$\mathcal{F} = M^{-3} \sum_{\mathbf{k}_0} \sum_{\mathbf{b}} \int d^3x_1 |\widehat{\mathbf{F}}_{\mathrm{gg}}(\mathbf{k}_0 + M\mathbf{b}) - \widehat{\mathbf{F}}(\mathbf{k}_0 + M\mathbf{b})|^2 \,. \tag{3.16}$$

Finding the optimal Green function for a general force field is analogous to the case with PM-forces, and is most easily done using the earlier result for PM-forces. The required force law for the PM-force is defined by the force of gravitational interaction of two S_2-shaped spheres [31], whose centers are separated by distance $|\mathbf{r}|$. The force field therefore obeys the Poisson equation and its Fourier transform is

$$\widehat{\mathbf{F}}_{\mathrm{S2}}(\mathbf{k}) = \frac{2iM\mathbf{k}}{k^2} |W_r(k, \eta)|^2 e^{-i2\pi\mathbf{k}\cdot\mathbf{x}_1/M} \,. \tag{3.17}$$

where we used a translation theorem to get the Fourier component of the force with the source point at $\mathbf{x}_1$ from its expression with the source in the center of coordinates, and the expression $W_r(k, \eta)$ for the Fourier transform of the density field of the S_2-

shaped sphere is given in [23]. For a general required interparticle force law, not necessarily obeying the Poisson equation as in the case of PP-force law, the Fourier component of the force field generated by a particle at the center of coordinates

$$\widehat{\mathbf{F}}_\phi(\mathbf{k}) = -i(\nabla\phi)_\mathbf{k} = -i\frac{2\pi\mathbf{k}}{M}\,\widehat{\phi}_\mathbf{k}\,e^{-i2\pi\mathbf{k}\cdot\mathbf{x}_1/M}\,, \tag{3.18}$$

where $\widehat{\phi}_\mathbf{k}$ is the Fourier transform of the potential field ϕ of a particle at the center of coordinate system, given by the required interparticle FPM-force law.

Extremizing (3.16) using $\widehat{\phi}_\mathbf{k}$ yields the only unknown coefficient $\mathbf{A}(\mathbf{k}_0)$ in the expression (A.14) of [23] for the Green function for computing the potential forces

$$\mathbf{A}(\mathbf{k}_0) = -\sum_\mathbf{b}(\mathbf{k}_0 + \mathbf{b}M)\frac{\pi\widehat{\phi}_\mathbf{k}}{M^2}\widehat{W}^2(\mathbf{k}_0 + \mathbf{b}M)\,. \tag{3.19}$$

We have derived at this point the optimal Green function for the force law of a general profile. Note that this derivation is only valid for the particles having compact, spherically symmetric density distribution. The potential field resulting from this distribution is compact, allowing us to use it for the FPM force.

In order to get the Fourier components of the potential field generated by a particle given the required FPM-force law $\Theta^0_{\rm FPM}$, the potential is first tabulated as a function of separation and is then sampled by the *potential mesh* gridpoints within the simulation volume, as shown in Figure 3-5. The Fast Fourier Transform of the sampled potential is then applied to the mesh in order to find the Fourier transform $\widehat{\phi}_\mathbf{k}$ of the true interparticle potential field. The potential cloud is limited by $\tilde{R}_{\rm c\,max}$ in the position space. When convolved with the Green function in Fourier space by Eq. (3.13), it yields the computed FPM-force field $\Theta_{\rm FPM}(r)$ of precisely the same spatial extent as the required FPM-force field $\Theta^0_{\rm FPM}(r)$ which has a cutoff separation $R_{\rm c\,max}$), $\Theta_{\rm FPM}(r > R_{\rm c\,max}) = 0$. This does not apply for the PM-forces since they are not spatially compact.

The fineness of the potential grid is set by the desired number of aliases to include in the sum Eq. (3.19), where the variable $\mathbf{b}$ ranges from $-l^{\rm FPM}_{\rm max}$ to $l^{\rm FPM}_{\rm max}$. By increasing the parameter $l^{\rm FPM}_{\rm max}$ we include more aliases and better sample the continuous potential field, which leads to better precision. In practice, the resolution parameter $l^{\rm FPM}_{\rm max}$ of the potential mesh is limited by the machine memory requirement constraint. We found, however, that increasing $l^{\rm FPM}_{\rm max}$ from zero does not lead to any significant improvement in force accuracy at least when the the required FPM-force law $\Theta^0_{\rm FPM}$ potential is smooth enough.

3.2.4 Force Error Analysis and Free Parameters

The free parameters for force calculation should be chosen on the basis of the required force accuracy and sometimes the available computing resources. In Table 3.1 we list all the free parameters for each force calculation scheme.

In this section, we perform a number of test simulations in order to find the optimal

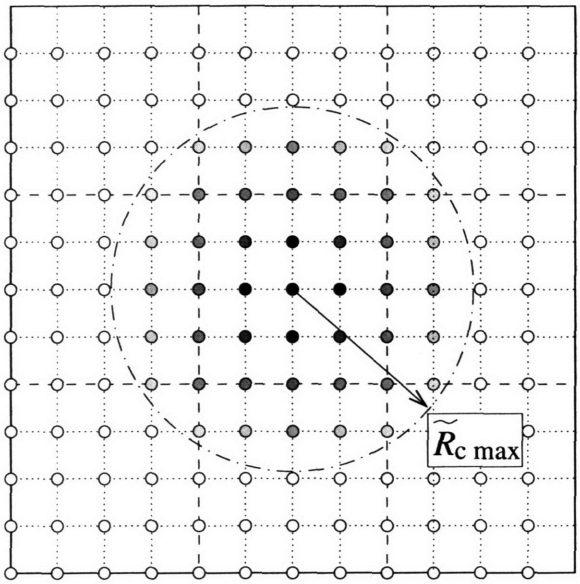

Figure 3-5: Sampling the potential given by the required FPM-force law with the potential mesh gridpoints (the circles) within the fine mesh simulation volume $B^{\mathrm{APP}}(r)$ in order to find the discrete approximation $\widehat{\phi}_{\mathbf{k}}$ to the Fourier components of the true continuous potential field limited by the short range cutoff distance $\tilde{R}_{\mathrm{c\,max}}$. Shading level shows the potential value. The FFT of the potential assignment gives $\widehat{\phi}_{\mathbf{k}} e^{-i\mathbf{k}\cdot\mathbf{x}}$, where the exponent is due to the shift of the field from the center of coordinates. Sampling with a finer potential mesh (choosing different $l_{\mathrm{max}}^{\mathrm{FM}}$) in general changes the spatial force resolution and is equivalent to a change in the number of aliases included within the sum in Eq. (3.19). The fine density mesh size is $n_{\mathrm{f}} = 4$. The resolution potential mesh resolution parameter is set to $l_{\mathrm{max}} = 1$, which means the size of the potential mesh (whose cells are shown by the dotted lines) is $(2l_{\mathrm{max}} + 1)n_{\mathrm{f}} = 12$. The boundaries of coarse mesh cells (the dashed lines) are irrelevant for the fine mesh geometry.

values for the free parameters given the required force precision. All the force accuracy simulations follow the same pattern. The particle content consists of one massive particle placed within the simulation box at a random position and a number N of test particles having negligibly small total mass and placed in random directions from the massive particle while sampling uniformly the logarithmic range of separations within $[\tilde{r}_{\mathrm{min}}, \tilde{r}_{\mathrm{max}}]$. A number of Monte Carlo simulations is performed with different particle distributions to get better statistics on force at a given separation.

We will focus predominantly on *random* and *systematic* force errors defined as functions of separation $\tilde{r}_0$ as

$$E_{\mathrm{RAN}}(\tilde{r}_0) \equiv \langle (\Theta - \langle \Theta \rangle)^2 \rangle^{1/2} \quad \text{and} \quad E_{\mathrm{SYS}}(\tilde{r}_0) \equiv \langle (\langle \Theta \rangle - \Theta_0)^2 \rangle^{1/2}, \tag{3.20}$$

where Θ is the measured force acting on a test particle placed in a random direction at separation $\tilde{r}_0$ from the randomly placed massive particle, Θ_0 is the exact force

Pure PM forces	
n^i	PM-density gridsize (n^i=128)
$l_{\max}^{\mathrm{PM}}$	Extra aliases to include into sum in Eq. (3.17), (2)
$\tilde{\eta}$	Size of S_2-shaped cloud (3.3)
PP forces	
$\tilde{\epsilon}$	Plummer softening length (0.1)
N_{pptab}	Size of the PP-force table (20001)
$\tilde{R}_{\mathrm{c\,max}}$	Short range force cutoff distance
FPM and FPP forces	
n_{f}	Fine density mesh size
C_{f}	See Eq. (3.8)
$l_{\max}^{\mathrm{FPM}}$	Extra aliases to include into sum in Eq. (3.19)

Table 3.1: Free parameters for force computation schemes. The bracketed numbers show the initial setting for our tests.

value at this separation, and the averaging is performed over the sets of Monte Carlo simulations, each having a test particle at any given separation of the tested sample of separations $\tilde{r}_0$. The quadrature sum of these errors identically gives the *absolute* error

$$E_{\mathrm{ABS}}(\tilde{r}_0) \equiv \langle(\Theta - \Theta_0)^2\rangle^{1/2} . \tag{3.21}$$

The systematic force errors can always be eliminated by tabulation. The random errors however are not reducible without a change in the algorithm or simulation parameters.

The ability to directly measure the random component of force error $E_{\mathrm{RAN}}(\tilde{r}_0)$ from Monte Carlo simulation is limited at small separations due to the effect of roundoff errors. Indeed, in order to find $E_{\mathrm{RAN}}(\tilde{r}_0)$ from the set of Monte Carlo simulation, one needs to average between many force calculations at a precisely given separation $\tilde{r}_0$ but random directions. Our ability to place the test particle at a given separation from the massive particle is limited by the effect of roundoff errors in coordinates. Indeed, the relative precision in floating point operations is limited by the sixth significant number. Since the positions of the massive particle are random within the box, the typical errors in the coordinates of test particles along each dimension will be $\delta\tilde{r}^i \approx 10^{-6} \max(\tilde{L}^i) = 10^{-6} L$ leading to the relative separation errors of

$$\frac{\delta\tilde{r}}{\tilde{r}} \approx \frac{\max(\delta\tilde{r})}{\tilde{r}} = \frac{10^{-6}\max(\tilde{L})}{\tilde{r}} . \tag{3.22}$$

At the separations well below the force softening length the Plummer force acting on a test particle grows proportionally to the separation. The value of the Plummer force at small separation is therefore uncertain with the same relative error as the error in separation and is diverging at small separations, according to the above equation. This uncertainty will also be reflected in the computed value of $\langle\Theta\rangle$ at small separations in equation (3.20), leading to the divergence in the computed sample in

Monte Carlo simulations

$$\frac{E_{\rm RAN}(\tilde{r}_0)}{|\Theta|} \geq \frac{10^{-6}\max(\tilde{L})}{\tilde{r}} \tag{3.23}$$

at small separations. In addition, by similar argument for the systematic errors we have

$$\frac{E_{\rm SYS}(\tilde{r}_0)}{|\Theta|} \propto \frac{10^{-6}\max(\tilde{L})}{\tilde{r}} . \tag{3.24}$$

The divergences at small separations in the above equations is the effect of numerical resolution of particle position while computing $\langle\Theta\rangle$ in Equation (3.20) for a given $\tilde{r}_0$. The divergence is not caused by the inaccuracy in the P^{3}M force calculation.

The P^{3}M force absolute error given by equation (3.21) does not contain the term $\langle\Theta\rangle$. We eliminate the effect of finite numerical resolution of position, leaving only the force errors, by modifying the use of equation (3.21). When a test particle is placed at a desired separation $\tilde{r}_0$ and random direction in a Monte Carlo simulation, its numerical separation $\tilde{r}_0'$ is different due to the roundoff errors so that $|\tilde{r}_0 - \tilde{r}_0'| \leq \delta\tilde{r}$, where $\delta\tilde{r}$ is approximately given by equation (3.22). To find $E_{\rm ABS}(\tilde{r}_0)$ as a function of $\tilde{r}_0$ we use the modified separation $\tilde{r}_0'$ in the right-hand side of equation (3.21), thus eliminating the effect of finite precision in separation. Note that once this recipe is used the true random and systematic force error components corrected due to the finite position resolution at a given separation $\tilde{r}_0$ are limited from above by the computed value of $E_{\rm ABS}(\tilde{r}_0)$.

Below, we start by analyzing the PM-force errors, proceeding with the P^{3}M and concluding with the adaptive P^{3}M-force errors. Unless noted otherwise, the simulation parameters in this section are given by the bracketed values in Table 3.1.

3.2.5 PM- and P^{3}M-force Error Analysis

In Figure 3-6 we present the average measured pure PM-force law with a few choices for $\tilde{\eta}$ parameter. The *required PM-force law* is set to the force law between two S_2-spheres, whose spatial extent is given by the $\tilde{\eta}$ parameter. From the figure we see that the measurements systematically deviate from the required PM-force law and the systematic error decreases with increasing $\tilde{\eta}$. The same is true for the random relative deviations $\langle(\Theta_{\rm PM} - \langle\Theta_{\rm PM}\rangle)^2\rangle/\langle\Theta_{\rm PM}\rangle$, shown in Figure 3-7.

We observed a very weak dependence of the measured systematic and average PM-force errors on the other parameters in Table 3.1. Changing the parameters n^i from 128 to 32 results in less than 0.1% change in the average measured force. The only point in increasing n^i in a P^{3}M simulation can be increasing spatial resolution of PM-forces, which means reducing the coarse density mesh and therefore coarse chaining mesh cell spacing as required for direct summation. Changing $l_{\rm max}^{\rm PM}$ between 0 and 2 changes the force law by less than 0.05% within the whole separation range. However we leave $l_{\rm max}^{\rm PM} = 2$ as a default value for now.

The systematic errors are eliminated in P^{3}M simulations by using the PM-force table defined below in order to sample values of $\langle\Theta_{\rm PM}\rangle$ that include the systematic

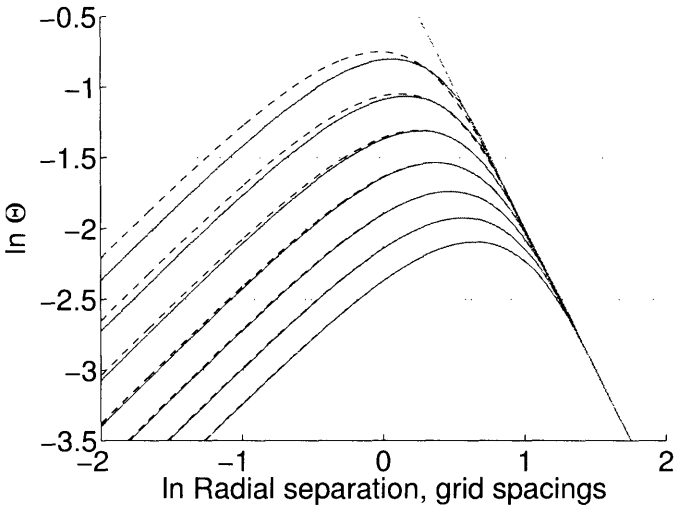

Figure 3-6: The required PM-force law $\Theta^0_{\rm PM}$ (*dashed lines*) and the force measurements $\langle\Theta_{\rm PM}\rangle$ (the solid curves) obtained by averaging over 1000 Monte Carlo simulations for different choices of S_2-sphere parameter $\tilde{\eta}$. Each point of the solid line is the measurement of the average parallel force component on a test particle at a given separation and $\tilde{\eta}$. The deviation between the dotted and the dashed curves show the systematic error in PM-forces $\langle\Theta_{\rm PM}\rangle - \Theta^0_{\rm PM}$. In the order of decreasing maxima of the curves, the parameters are $\tilde{\eta} = 2.5, 2.9, 3.3, 3.7, 4.1, 4.5, 4.9$. The straight line shows the inverse square force law.

errors as a function of separation.

The PP-force is introduced in P^{3}M simulations as a radial correction for the computed PM-force $\boldsymbol{\Theta}_{\rm PM}(\tilde{\mathbf{r}})$ needed in order to bring the average value of the force (including the systematic error contribution) to the the required P^{3}M Plummer interparticle force law

$$\Theta_{\rm PP}(\tilde{r}) \equiv \Theta_{\rm PL}(\tilde{r}) - \Theta^{||}_{\rm PM}(\tilde{r}) \,, \tag{3.25}$$

where $\Theta^{||}_{\rm PM}(\tilde{r}) \equiv \tilde{\mathbf{r}} \cdot \langle\boldsymbol{\Theta}_{\rm PM}\rangle(\tilde{\mathbf{r}})/\tilde{r}$ is the parallel component of the average PM-force law, found by tabulating a large sample of test particles in a PM-test simulation at separations $\tilde{r}_i$. The *PM-force table* $\Theta^{\rm T}_{\rm PM}(\tilde{r}_i)$ is defined as $\Theta^{||}_{\rm PM}(\tilde{r}_i)/4\pi r_i$ in order to avoid roundoff inaccuracy at small separations, where the $\Theta^{||}_{\rm PM}(\tilde{r})$ reaches zero, but $\Theta^{||}_{\rm PM}(\tilde{r}_i)/4\pi r_i$ achieves a constant value suitable for interpolation.

The P^{3}M force computed using the force table and equation (3.10) at one of the tabulated separations $\tilde{r}_i$ does not have systematic errors due to the systematic difference of the computed PM-force law from the required PM-force law. The systematic error in radial force component is eliminated by introducing the PM-table, and there appears to be no systematic errors in the tangential component. For the separations between the tabulated values r_i and r_{i+1} the P^{3}M force is computed using linear interpolation between the force table values $\Theta^{\rm T}_{\rm PM}(\tilde{r}_i)$ and $\Theta^{\rm T}_{\rm PM}(\tilde{r}_{i+1})$.

Even when the number of test particles used in the PM-test simulation is huge,

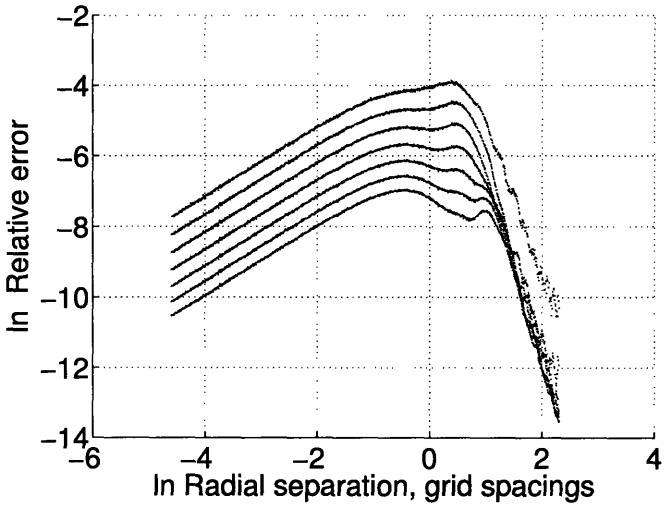

Figure 3-7: Relative random error of parallel PM-force component $\langle(\Theta_{\rm PM} - \langle\Theta_{\rm PM}\rangle)^2\rangle^{1/2}/\Theta^0_{\rm PM}$ for the same simulations as Figure 3-6. The parameters $\tilde{\eta}$ increase in the order of decreasing curve maxima.

the computed values of PM-table $\Theta^T_{\rm PM}(\tilde{r}_i)$ are noisy. As we know from Sections 3.2.2 and 3.2.3 [see also equation (3.18)], in the adaptive P^{3}M, the Fourier Transform of the PP-force law $\Theta_{\rm PP}(\tilde{r})$ is used in order to define the FPM forces. Using Fourier Transformation on a noisy or discontinuous field would result in an oscillating behavior for the computed FPM forces. In order to eliminate these artificial fluctuations which would otherwise yield systematic errors in the computed adaptive-PP forces, we: a) smooth the fluctuations out by sparsely sampling the existing PM-table with a few points covering the whole PM-table separation range and obtaining the stable PM-table values for those few points by doing a boxcar average of the PM-table values in their vicinity and doing a concavity correction; and b) adjust the PP-force cutoff separation $\tilde{R}_{\rm c\,max}$ so that the PP-force at that separation, as determined by equation (3.25), is exactly zero making the PP-force continuous at the PP-force cutoff separation $\tilde{R}_{\rm c\,max}$. The resulting PP-force will therefore be a function of the required P^{3}M force law (the Plummer force law) and the parameter $\tilde{\eta}$ used to setup the PM-forces. The parameter $\tilde{R}_{\rm c\,max}$ is a function of $\tilde{\epsilon}, \tilde{\eta}$

$$\tilde{R}_{\rm c\,max} = \tilde{R}_{\rm c\,max}(\tilde{\epsilon}, \tilde{\eta}) \tag{3.26}$$

determined by conditions a) and b) above.

Using the combination of equations (3.10) and (3.25), using the interpolation from PM-table to get the values of $\Theta^{||}_{\rm PM}(\tilde{r})$ and evaluating the Plummer force law by equation (3.11) is impractical, due to the relatively high cost of the Plummer force evaluation and high number of direct summations for PP-forces in simulations. In order to avoid Plummer force function evaluation in a P^{3}M simulation, the PP-force law is pretabulated at $N_{\rm pptab}$ densely placed points at the squared separations given

by

$$\tilde{r}_i^2 = i\tilde{R}_{\mathrm{c\,max}}^2/(N_{\mathrm{pptab}} - 1) \, .$$

By using this sampling rule it takes only a few floating point operations to find the PP-table index i given any pairwise separation $r \leq \tilde{R}_{\mathrm{c\,max}}$ and therefore the PP-force evaluation at any separation $\tilde{r} \leq \tilde{R}_{\mathrm{c\,max}}$. This sampling rule also provides dense sampling at the separations close to the cutoff distance $\tilde{R}_{\mathrm{c\,max}}$.

When the Plummer softening length is set to a sufficiently small value, fair sampling of Plummer force by PP-table is not reached at small separations. In order to avoid this problem one can either impose a constraint on $\tilde{\epsilon}$

$$\tilde{\epsilon} \geq \frac{C_\epsilon \tilde{R}_{\mathrm{c\,max}}}{\sqrt{N_{\mathrm{pptab}} - 1}} \, , \tag{3.27}$$

where the parameter $C_\epsilon \gtrsim 1$ or use Plummer force evaluations directly by equation (3.11), instead of using the PP-force table at the small separations.

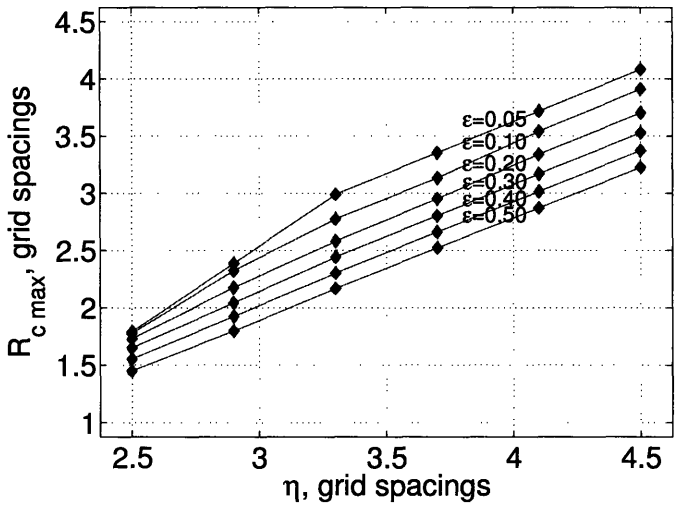

Figure 3-8: The dependency of $\tilde{R}_{\mathrm{c\,max}}$ on $\tilde{\epsilon}$ and $\tilde{\eta}$ for the case of the Plummer force law. The values of $\tilde{\epsilon}$ are shown above the curves. This plot is not used for evaluating $\tilde{R}_{\mathrm{c\,max}}$, instead it is only an illustration giving the value of $\tilde{R}_{\mathrm{c\,max}}$ for various $\tilde{\epsilon}$ and $\tilde{\eta}$, produced by the procedure leading to equation (3.26).

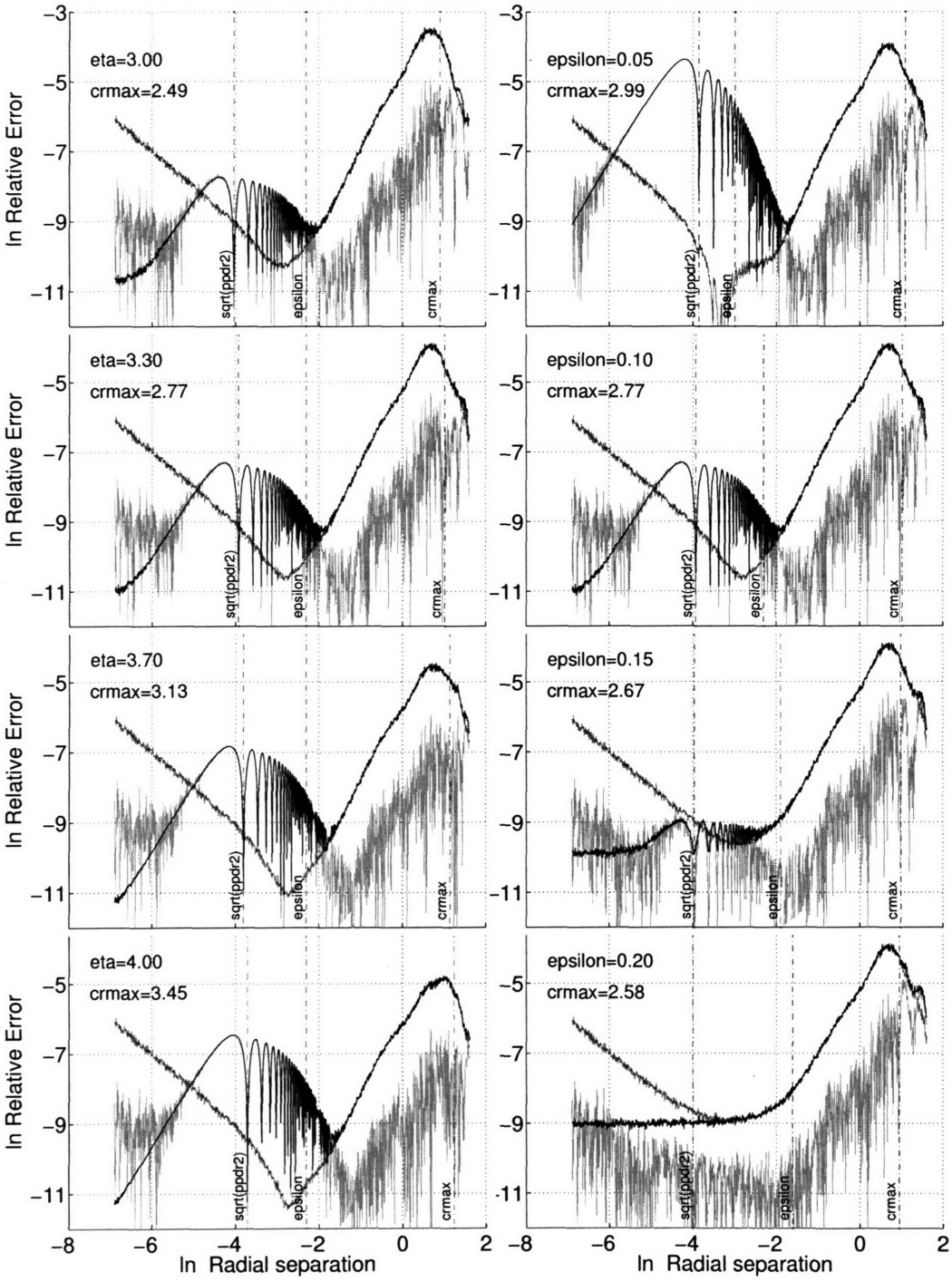

Figure 3-9: Non-adaptive P^{3}M. Simulation testing errors as function of $\tilde{\eta}$ and $\tilde{\epsilon}$. Left: variation of $\tilde{\eta}$ at a fixed $\tilde{\epsilon} = 0.1$ The relative errors for some values of $\tilde{\eta}$ reach e^{-4} near $\tilde{R}_{\mathrm{c\,max}}$. The $\tilde{\eta}$ parameter of the PM-force simulation controls the maximum magnitude of this error. Right: variation of $\tilde{\epsilon}$ at a fixed $\tilde{\eta} = 3.3$. Each plot is the result of 100 Monte Carlo simulations. In order of decreasing shading, the curves show $E_{\mathrm{ABS}}(\tilde{r}_0)/\Theta^0_{\mathrm{P3M}}$, $E_{\mathrm{RAN}}(\tilde{r}_0)/\Theta^0_{\mathrm{P3M}}$ and $E_{\mathrm{SYS}}(\tilde{r}_0)/\Theta^0_{\mathrm{P3M}}$. The value of $\tilde{R}_{\mathrm{c\,max}}$ is given on each panel as crmax. The measured values $E_{\mathrm{RAN}}(\tilde{r}_0)/\Theta^0_{\mathrm{P3M}}$ and $E_{\mathrm{SYS}}(\tilde{r}_0)/\Theta^0_{\mathrm{P3M}}$ increase at small separations due to the numerical resolution of particle position, see Section (3.2.4).

Let us do the full force P^3M simulations now. We adopted the value $N_{\text{pptab}} = 20001$ and plot the P^3M random, systematic, and absolute relative force errors at different $\tilde{\eta}$ and $\tilde{\epsilon}$ in Figure 3-9 for the test particle separation range $\tilde{r} \in [0.001, 5]$ which includes $\tilde{R}_{\text{c max}}$ within the range. The divergence of relative random force errors at small separations is due to equation (3.23). In the following we will limit our discussion to the accuracy in P^3M force computation, for which the relevant plotted value at small separations is $E_{\text{ABS}}(\tilde{r}_0)$, as explained in Section 3.2.4.

In agreement from our expectations from pure PM (Figure 3-7), the simulations show the increase of force accuracy with the increase of the S_2-cloud shape $\tilde{\eta}$. From the measurements from various $\tilde{\epsilon}$, we see that the optimal value of parameter C_ϵ such that the errors resulting from poor PP-force tabulation at small separation are below 0.2% is $C_\epsilon \gtrsim 3$.. This parameter is automatically checked at the start of the run.

The force errors increase sharply at small $\tilde{\epsilon}$, as seen from Figure 3-9, as $\tilde{\epsilon}$ drops below 0.05, the errors increase above the 1.5% level due to poor sampling of the Plummer force law at these $\tilde{\epsilon}$. At this point, the rate of increase is so high that a simulation with $\tilde{\epsilon}$ just below 0.05 leads to unacceptable force errors. Although it is possible to use a different interpolation scheme or even direct force evaluation at small $\tilde{\epsilon}$, this procedure is not implemented for the purpose of this Thesis, where higher values of $\tilde{\epsilon}$ are sufficient. We leave this code improvement for future work.

3.2.6 FPM- Required Force Law

As we discussed in Section 3.2.3, the FPM-force uses an interparticle short range potential Eq. (3.18) and the FFT to get the PP-force, similar to the way the PM-force uses the analytical Fourier transforms of the S_2-sphere to get the PM-force. In our experiments, we found that using PP-force alone as the required force law leads to unacceptable random and systematic errors in the computed total adaptive P^3M force. To avoid them, the required FPM-force law should satisfy certain smoothness and locality constraints. It is necessary to modify the required force law at short ranges so that the random errors in the computed FPM-force are small.

The adaptive P^3M force errors are very sensitive to the required FPM-force law at short ranges. Using direct testing of different PP-force profile short range corrections we found that the force law given by the first two even powers of the separation $\tilde{r}$

$$\Theta^0_{\text{FPM}}(\tilde{r}) \equiv \begin{cases} A + B\tilde{r}^2 + C\tilde{r}^4 & \tilde{r} < \tilde{R}_{\text{f max}} \\ \Theta_{\text{PP}}(\tilde{r}) & \tilde{r} \geq \tilde{R}_{\text{f max}} \end{cases} \tag{3.28}$$

with matched first and second derivatives on both sides of the transition point $\tilde{r}_0 = \tilde{R}_{\text{f max}}$

$$\begin{aligned} A &= \Theta_{\text{PP}}(\tilde{r}_0) - \tfrac{5}{8}\tilde{r}_0\Theta'_{\text{PP}}(\tilde{r}_0) + \tfrac{1}{8}\tilde{r}_0^2\Theta''_{\text{PP}}(\tilde{r}_0) \\ B &= -\tfrac{1}{4}\Theta''_{\text{PP}}(\tilde{r}_0) + \tfrac{3}{4}\tilde{r}_0^{-1}\Theta'_{\text{PP}}(\tilde{r}_0) \\ C &= \tfrac{1}{8}\tilde{r}_0^{-2}\Theta''_{\text{PP}}(\tilde{r}_0) - \tfrac{1}{8}\tilde{r}_0^{-3}\Theta'_{\text{PP}}(\tilde{r}_0) \end{aligned}$$

result in small total force errors. The required force law Θ^0_{FPM} including this short range correction is presented in Figure 3-2 for illustration.

Now, we can use the equation similar to Eq. (3.25) to define the FPP-forces. In contrast to the PM-forces, using the profile in Eq (3.28) results in small systematic errors; that is, we have to a very high accuracy $\langle \Theta_{\rm FPM} \rangle = \Theta^0_{\rm FPM}$. Absence of the systematic errors allows us to avoid the tabulation of forces. We define the FPP-force by

$$\Theta_{\rm FPP}(\tilde{r}) \equiv \Theta_{\rm PP}(\tilde{r}) - \Theta^0_{\rm FPM}(\tilde{r}) \, , \tag{3.29}$$

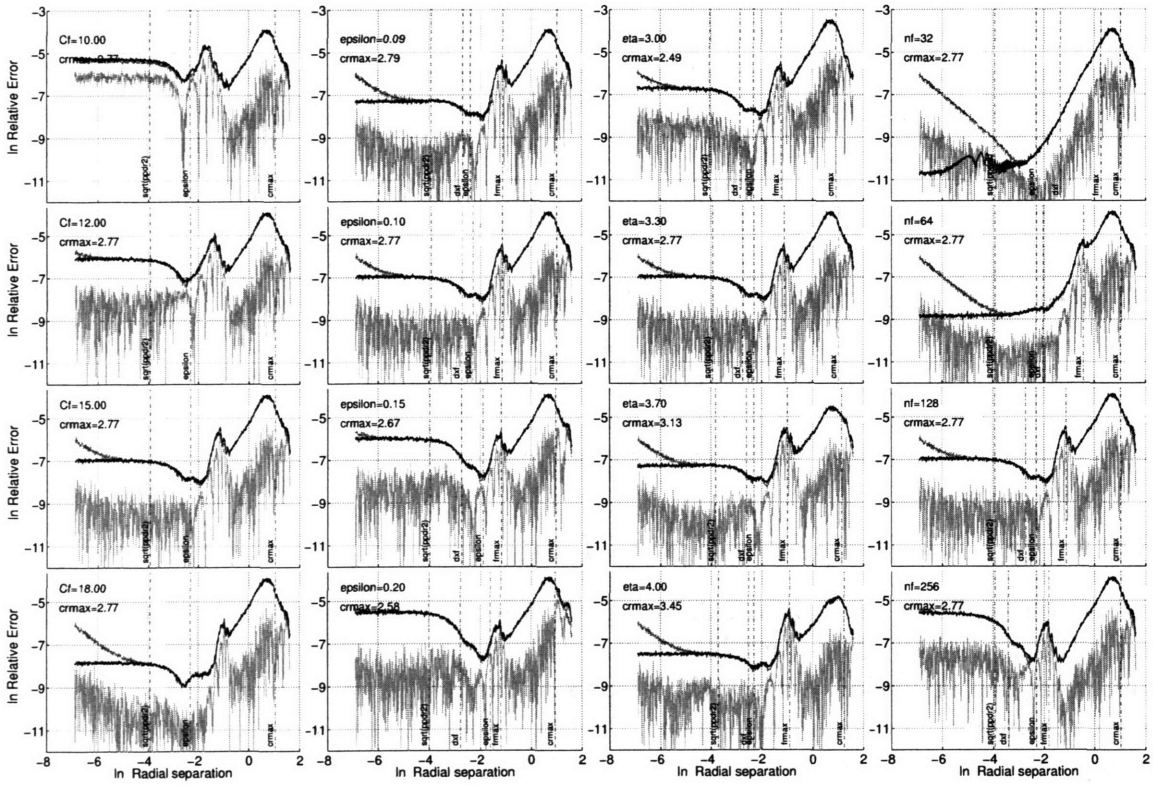

Figure 3-10: Adaptive P^{3}M-force relative errors, for various free simulation parameter choices. The measured values $E_{\rm RAN}(\tilde{r}_0)/\Theta^0_{\rm P3M}$ and $E_{\rm SYS}(\tilde{r}_0)/\Theta^0_{\rm P3M}$ increase at small separations due to the numerical resolution of particle position, see Section (3.2.4).

To analyze the resulting random and systematic total adaptive P^{3}M force errors, as well as their dependence on simulation parameters, we performed a set of Monte Carlo simulation tests presented in Figure 3-10. At any of the parameters we tested, the systematic error is always less than the random force errors. In addition the absolute relative errors have a flat profile at small separations and in general are acceptably small. These properties established our choice for the required FPM-force law given by Eq. (3.28).

We observe that the errors grow at the short range with the decrease of $C_{\rm f}$. On the other hand, the increase in $C_{\rm f}$ results in higher FPP-direct summation cost since the fine chaining mesh cell spacing increases. The optimal value of $C_{\rm f}$, achieved as a

compromise between these two effects is

$$C_{\rm f} = 15\,. \tag{3.30}$$

The tested values of $\tilde{\epsilon}$, constrained by the range given by Eq. (3.27) and $\tilde{\epsilon} \leq 0.1\tilde{R}_{\rm c\,max}$ do not result in significant errors.

The choice for $\tilde{\eta}$ determines the main contributions to the errors coming from the maximum of the error curves at $\tilde{r}$ just below $\tilde{R}_{\rm c\,max}$. One should choose higher $\tilde{\eta}$ for higher accuracy. Note however that the resulting value for $\tilde{R}_{\rm c\,max}$, given by Eq. (3.26) increases linearly with $\tilde{\eta}$, resulting in more direct summations for PP-force calculation, and lower resolution of HC-mesh.

With an increase in the parameter $n_{\rm f}$, the random component of force error at small sepaparation increases, exceeding the systematic error component by a large factor. The random errors can not be eliminated by a different use of the force tables. They are entirely due to the errors in the FPM force evaluation. Within a wide range of $n_{\rm f}$ shown in figure, the force errors do not exceed $\exp(-5.3) \approx 0.45\%$, increasing at small ranges by approximately a factor of 5 each time the parameter $n_{\rm f}$ doubles. By extrapolation, the relative force errors are reaching 0.5% at $n_{\rm f} = 288$ and 1.6% at $n_{\rm f} = 448$.

The increase of $n_{\rm f}$ leads to higher resolution of the fine density mesh and therefore higher cost of doing FFT for FPM-force calculation. On the other hand, by increasing $n_{\rm f}$ we decrease the cost of the direct summation in FPP, because the parameter $\tilde{R}_{\rm f\,max}$ is lower at higher $n_{\rm f}$. The optimal value of $n_{\rm f}$ should be chosen individually for each cell for which the mesh refinement is performed. The value depends on the balance of the FPM-force and FPP-force calculation loads. If it takes too much time to do FPP-direct summation in comparison with the time it takes to do FPM, the parameter $n_{\rm f}$ should be increased. This will lead to smaller spacings within the fine chaining mesh and eventually lead to transforming some of the FPP-load into FPM. If $n_{\rm f}$ is set to the optimal value, the FPM and FPP-loads should be comparable. In the following sections we show how to set the value of $n_{\rm f}$ optimally.

3.3 Total Workload Minimization

As discussed in Chapter 2, for each process i the workload of advancing the N-body system one timestep, equal by definition to the wall clock time $W(i)$ spent by the process i on this timestep, can be decomposed into two parts

$$W(i) = W_{\rm wait}(i) + W_{\rm cpu}(i) \tag{3.31}$$

where $W_{\rm wait}(i)$ is the waiting time and $W_{\rm cpu}(i)$ is the CPU time spent by the process i on this timestep. The waiting time $W_0(i)$ can only be spent waiting for interprocess communications to finish (although it can also be due to excessive paging, see Section 2.4.1 for more discussion) and therefore can not be attributed exclusively to some specific cells of the local domain. In contrast, the CPU Time $W_{\rm cpu}(i)$ can be subdivided into two parts: one $W_{\rm HC}(i)$ that can be directly associated with processing

each of the local HC-cells, and the other $W_{\rm NL}(i)$ which is not

$$W_{\rm cpu}(i) = W_{\rm HC}(i) + W_{\rm NL}(i) \tag{3.32}$$

The load balancing technique presented in Chapter 2 is designed to load balance only the cell specific CPU workloads $W_{\rm HC}(i)$ of each process since this part can be controlled by repartitioning. Since the cell specific workload, which is due mainly to the PP-direct summation, constituted the major part of the total workload, by load balancing them we have load balanced the total CPU workloads.

The adaptive force computation technique introduced in this Chapter is designed to reduce the cell specific CPU workload for the short range force computation. If this technique is efficient enough the fraction of the non-cell specific CPU workloads may become significant. In this section, we describe the methods used in order to minimize the total workload: both waiting time and CPU cell specific workload for short range force computation.

In Section 3.3.2 we show how we reduce the PM-computation waiting time workload and show that the reduction is significant. In Sections 3.3.2 and 3.3.3 we present the method used in order to select the optimal mesh refinement gridsize of each refined HC-cell in order to minimize the adaptive PP-force computation cell specific workloads. In Section 3.3.4 we show how by using a new sorting technique we are able to significantly reduce the FPP cell workloads.

3.3.1 Non-blocking PM-communications

In this section we first demonstrate that due to significant waiting time good scalability can not be achieved for PM-force computation using blocking MPI communications with our domain decomposition. We then introduce an improvement of the original PM-force computation algorithm described in Section 2.6.1 that reduces the waiting time cost of computing PM-forces, or the $W_{\rm wait}$ term in equation (3.31).

The PM-force computation performed every timestep requires the exchange of large amounts of data (density and force messages) by the processes between completely different domains. In Section 2.6.1 we presented the efficient parallel algorithm for parallel computing of PM-forces. The exchange of data between the processes i and j occurs once for each non-empty term in either of the two sets

$$G^j_{\rm sl} \cap \mathcal{R}(M^{ik}_h), \quad k = 0 \ldots n^i_k - 1$$

or

$$G^i_{\rm sl} \cap \mathcal{R}(M^{jk}_h), \quad k = 0 \ldots n^j_k - 1$$

as defined in equation (2.42).

As the number of processes grows the number of messages per process required for PM-force computation the number of PM-grid messages required grows too. Indeed, we can get the lower limit for this number by the number of non-empty terms in the sum $\sum_{j=0}^{n_{\rm pr}-1} G^j_{\rm sl} \cap L^i_h$. Considering low clustering regime of matter distribution for simplicity, where the particle domains L^i_h of each process occupy the volume $N_{\rm HC}/n_{\rm pr}$

HC cells, we find that due to the compactness property of the Hilbert curve all local regions occupy quasi-spherical volumes of radius given in code units by

$$\Delta\tilde{x}_h \left(\frac{3}{4\pi} \frac{N_{\mathrm{HC}}}{n_{\mathrm{pr}}} \right)^{1/3} .$$

The FFTW slabs G^j_{sl} on the other hand have planar geometry extending over the entire simulation volume in dimensions 1 and 2, while their thickness in dimension 0 is given in code units by n^0/n_{pr}. Comparing the extents of the domains we observe that at $n_{\mathrm{pr}} \geq 2$ the FFTW-slabs have small thickness compared to the extent of the particle domains.

Given the FFTW slab G^i_{sl} of process i, the number of local domains of processes j having a non-zero intersection $G^i_{\mathrm{sl}} \cap L^j_h$ is therefore evaluated by dividing the area of the FFTW-slab by the average area of the cross section of a local domain. We find that each FFTW-slab directly overlaps with $\approx n_{\mathrm{pr}}^{2/3}$ particle domains of other processes. On the other hand, one can evaluate the number of FFTW slabs G^j_{sl} of processes j receiving a non-zero contribution from the local domain L^i_h of process i by dividing the extent of the local domain by the thickness of the an FFTW-slab, which gives us another $\approx n_{\mathrm{pr}}^{2/3}$ communications required. All contributions add up to the total of $\approx 2n_{\mathrm{pr}}^{2/3}$ communications required on average of each process to exchange the density/force mesh between the local regions and the FFTW domains. In practice however the number of PM-messages is higher by a factor of few due to the finite TSC window function locality length, the disjoint local domains and the decomposition of local regions L^i_h into the slices M^{ik}_h along the 0-th dimension.

The average length of each message is evaluated by taking a product of the FFTW slab thickness and the area of the cross section of a local domain, both expressed in the PM density mesh spacings, giving in the result 4bytes $\times$ N_{gr} $n_{\mathrm{pr}}^{-5/3}$ per each message. In practice, due to the run length compression of PM messages as described in Section 2.6.1, the average size of the grid point messages is lower by a significant factor depending on the clustering of matter.

For a simulation with an 800^3 PM-grid performed on 80 processes and described in Section 2.7.3, the above estimates together with all the corrections yield roughly 150 messages for each process to exchange with the others during during Steps 1 and 5 of the PM-force computation. Each PM-grid message has the average size 1.3 MB divided by the compression factor which is of the order of unity when the matter distribution is close to uniform and reaches much higher values in the regime of strong clustering.

For all the test runs performed in Chapter 2 we used blocking MPI messaging for PM-force calculations, meaning that whenever the communication in the pair of processes i and j is processed, both processes have to wait until the communication is complete. By using blocking communications we can not achieve good scalability for PM, since for each process i the number of synchronizations required with other processes j is large and grows with the number of processes as $n_{\mathrm{pr}}^{2/3}$. Since the other processes are involved with their own communication requests, using blocking communications results in a significant amount of overall waiting time. Indeed, in

the long 800^3 run described in Chapter 2, the ratio of the average wall clock time to the average CPU time spent during the complete PM force evaluation evolved from 5.2 to 6.5 during the course of the run, meaning that most of the time during the PM calculation the CPUs were idle.

In the `llap3m-hc` code we are presenting in this Chapter, we reduce this ratio of the waiting time by implementing the so called non-blocking MPI communications for the PM-density and force messages. Non-blocking communications allow each ij pair of processes to go forward without waiting for the communications to clear. Non-blocking communications allow many communications to proceed simultaneously. Non-blocking communications do not require synchronization between processes, therefore they result in a significantly reduced waiting time.

There are two limitations for non-blocking communications. First, our experiments show that the number of simultaneously ongoing non-blocking communications is usually limited by an MPI implementation. For the test runs described later in Section 2.7 we set the limit to the number of communications ongoing simultaneously to 100. Whenever the number of ongoing communication requests exceeds this limit the processes start waiting for the requests to clear. This parameter is readily modified as allowed by the user's MPI implementation. Setting this parameter to 1 is equivalent to the return to the blocking communications.

Each of the ongoing communications requires a separate buffer of memory space to be filled as each of the communication requests completes. In the case of blocking communications, one needs to hold only the buffers sufficient for the currently processed communication. For the non-blocking communications, the storage for all the ongoing communications must be allocated simultaneously.

One can get an estimate for the upper bound for the memory required to send and receive the messages during the communications by using the above estimates for the average PM-grid message size and the number of PM-messages required. Multiplying the two numbers we get the maximum memory requirement

$$4\text{bytes} \times \frac{\text{few} \times N_{\text{gr}}}{\text{compression factor} \times n_{\text{pr}}}$$

valid at low clustering. This memory requirement is perfectly scalable, and it also shrinks shrinks in high clustering regime due to the rising compression factor for the PM-grid messages.

In addition to the message size there is yet another extra storage, associated with the local grids $\mathcal{R}(M_h^{ik})$ used for density interpolation from particles (here the M_h^{ik} is the k-th slice of the local region L_h^i dissected into n_k slices along the 0-th dimension as described in Section 2.6.2. This storage may potentially add up to a significant number for the runs where voids occupy a significant portion of the whole simulation volume such as the one described in Section 2.6.2. Since the total amount of memory allocated is tracked in our code the number of ongoing communications may be easily controlled depending their memory requirements. However even in our biggest runs performed in this section the memory remains very well balanced and we did not face any of the potential complications so far. We are leaving the implementation of

the memory based constraint on non-blocking communications for future, if it ever becomes vital.

Let us mention again that by introducing non-blocking communications for PM, we only reduced the portion of workload that is not attributed to specific HC-cell, rather it is attributed to the local domain as a whole. The PM-cell specific workloads $\tilde{w}^{\mathrm{PM}}(n)$ used to define the CPU cell workloads of processes by equation (2.24) are not influenced by this change.

3.3.2 The Workload Model

The simulations in Section 3.2.3 show that setting the mesh refinement parameter n_{f} to any value between $n_{\mathrm{f}} = 32$ and $n_{\mathrm{f}} = 448$ yields good force accuracy, with relative force errors below 1.6%. In this section we develop a model relation for the dependence of the workload of a cell on the mesh refinement parameter n_{f} and the number of particles in the cell. We assume uniform particle distribution over a few cell spacings to derive the model relation. Using this model relation we are able to find the optimal individual value for the parameter n_{f} for each mesh refinement, based on the known number of particles contained within the mesh refined cell by minimizing the expected adaptive-PP cell workload, entering the term W_{HC} of the equation (3.32) for total CPU workload for a process. In the following Section 3.3.3 we develop a practical scheme that uses these model relations and cell workload measurements but not the number of particles in a cell in order to find the optimal value for n_{f}.

Let us roughly evaluate the FPP workloads of a coarse chaining mesh cell cell w_{FPP} containing x particles and refined with a fine chaining mesh having c cells ($c \equiv n_{\mathrm{fc}}^3$). The parameter n_{f} influences the FPP workload through the dependence of the size n_{fc} of coarse chaining mesh (used by FPP to select close pairs of particles for direct summation) on n_{f} at a given fixed C_{f} [see equations (3.6) and (3.8)].

Assuming for simplicity a uniform particle distribution within the fine mesh simulation volume (containing the same coarse mesh cell plus the 26 surrounding cells), the fine mesh simulation volume contains $27x$ particles, and each fine chaining mesh cell contains has $27x/c$ particles. The spacing of the fine chaining mesh is roughly equal to the short range cutoff distance for FPP. While doing the FPP-force calculation, each particle is therefore involved on average into $(1/2 + 26) \times (27x/c)$ direct summations with the particles within the same and the surrounding cells. Multiplying this by the total number of particles $27x$ within the fine mesh simulation volume, we have the total $(1/2 + 26) \times ((27x)^2/c)$ for the number of the direct summations needed to perform the FPP-force computation for all the particles within the coarse mesh cell. Since it takes roughly the same amount of time needed for force update of each pair of particles, using B_2 as the overall coefficient of proportionality, we have the rough estimate for the FPP cell workload

$$w_{\mathrm{FPP}} \approx B_0 + B_2 \frac{x^2}{c} \,. \tag{3.33}$$

where B_0 is an additional constant contribution resulting from the overhead of doing

the FPP calculation.

In contrast to FPP, the main contribution into the FPM-workload comes from the cost of doing Fourier Transforms. There are $M_{\rm FPM}$ (currently $M_{\rm FPM} = 10$) Fourier transforms on the density mesh of size $c = n_{\rm f}^3$ required for each FPM force calculation. Let their total workload be by definition $A_2\, c \log c$, where A_2 is a number that in general depends on c. Setting the workload of doing one Fourier Transform to $A_{\rm FFT}(c)\, c \log c$ by definition, the coefficient A_2 is expressed in terms of $A_{\rm FFT}$ as

$$A_2(c) = M_{\rm FPM} A_{\rm FFT}(c)\,. \tag{3.34}$$

There is an additional (usually small) contribution to the FPM workload, proportional to the number of particles, due to the interpolation of mass and force between the particles and the fine density grid. Setting A_1 to be the coefficient of proportionality, and adding the constant A_0 we have in the total

$$w_{\rm FPM} \approx A_0 + A_1 x + A_2\, c \log\, c\,. \tag{3.35}$$

Note that the coefficients B_2 and A_1 in Eqs (3.33) and (3.35) are roughly related as $A_1 \approx M_{\rm FPM} B_2$, since the number of floating point operations and memory accesses involved into the FPM density and force evaluation is roughly $M_{\rm FPM}$ times that for FPP.

The total adaptive-PP workload necessarily includes both $w_{\rm FPM}$ and $w_{\rm FPP}$. However in general it may also include an additional term proportional to the number of particles $C_1\, x$. Adding all the contributions, and grouping all the linear terms, we have

$$w_{\rm APP} \equiv w_{\rm FPM} + w_{\rm FPP} = C + C_1 x + A_1 x + A_2\, c \log\, c + B_2 \frac{x^2}{c}\,. \tag{3.36}$$

where $C \equiv A_0 + B_0 + C_0$ is the overhead of doing adaptive-PP computation.

By differentiation $\partial_c w_{\rm APP} = 0$, and ignoring the dependence of A_2 on c, we find

$$x^2(c) = c^2 \frac{A_2(1 + \log c)}{B_2}\,. \tag{3.37}$$

This provides a linear relation for the optimal gridsize c as a function of the number of particles x. Using this gridsize for mesh refinement yields the minimum possible (the optimal) workload of the adaptive-PP computation.

It is interesting to note that equating the terms $w_{\rm FPM}$ and $w_{\rm FPP}$ in the limit $c \gg 1$ and (3.33) neglecting the usually small terms containing A_0, A_1 and B_0 we arrive to the same equation (3.35) in the limit of $c \gg 1$. We conclude from this observation that setting the chaining mesh gridsize c to a value that results in even distribution of the workloads between FPP and FPM leads to the minimal adaptive-PP cell workload.

Plugging equation (3.37) into (3.36) yields the function of the optimal adaptive workload on the optimal fine density gridsize c, or on the number of particles [through

equation (3.37)]

$$w_{\rm APP}(c) = C + \left[A_2 + (C_1 + A_1)\left(\frac{A_2(1+\log c)}{B_2}\right)^{1/2}\right] c + 2A_2 c \log c\,. \qquad (3.38)$$

The optimal workload varies almost linearly with the number of particles x per cell:

$$\frac{dw_{\rm APP}}{dx} = \frac{dw_{\rm APP}/dc}{dx/dc} = C_1 + A_1 + 2\sqrt{A_2 B_2(1+\log c)}\,, \qquad (3.39)$$

leading to the approximately linear dependence of the adaptive PP cell workload $w_{\rm APP}$ on the number of particles x within the fine mesh simulation volume

$$w_{\rm APP} \propto x\sqrt{\log x}\,, \qquad (3.40)$$

since $x \propto c\log c$ by equation (3.37).

Obviously, if we were to use a single choice of $n_{\rm f}$ for all mesh refinements within the simulation volume we would have a limited advantage from mesh refinement, since it would result in a quadratic dependence of the cell workload on the number of particles, as shown by Eq. (3.36) for constant c. If instead we use equation (3.37) to make a selection of the gridsize c *individually* for each mesh refined cell based on the number of particles x they occupy, we achieve the almost linear dependence of the cell workload on the number of particles with the linearity slope given by equation (3.37).

Equations (3.39) demonstrate that that minimizing the coefficients A_1, A_2, B_2, C_1 leads to the shallower slope of the workload as a function of the number of particles and therefore more effective adaptive mesh refinement in general.

Let us show now that using mesh refinement for low particle number coarse mesh cells does not lead to the optimal workload. Indeed, at the number of particles x decreases so does the optimal gridsize c [see equation (3.37)]. At small c, we can neglect all the terms in equation (3.38) except for the overhead cost C, arriving to $w_{\rm APP}(c) \approx C$. We can now compare this workload with the workload of doing the PP-force calculation by the direct summation, which can be evaluated by using the same arguments as those used to arrive to equation (3.33). In contrast to FPP, the PP forces are computed using only half of the 3^3 surrounding cells and involve no significant overhead. Using these arguments we find the expression

$$w_{\rm PP} \approx \frac{B_2 x^2}{2\cdot 3^3} \qquad (3.41)$$

for the cell-workload for PP-force computation by the direct summation. Comparing the two expressions, we find that using mesh refinement is not optimal when the number of particle within the cell satisfies

$$x < \sqrt{\frac{54C}{B_2}}\,. \qquad (3.42)$$

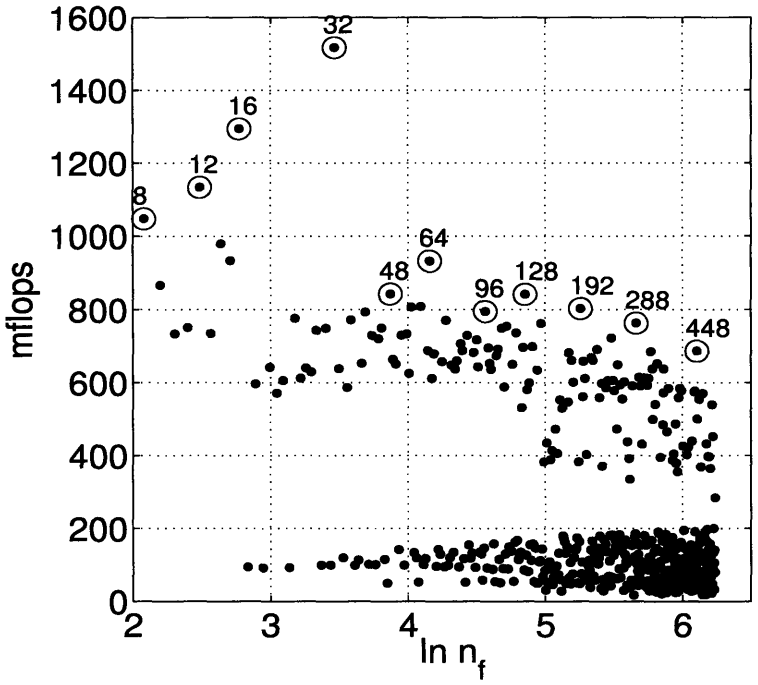

Figure 3-11: The plot of the FFTW speed expressed in Megaflops, as a function of the gridsize n_f used for a cubic grid Fourier transform. The FFT is performed on one process. The circled measurements are labeled by their n_f.

3.3.3 Workload Minimization Scheme

The coefficient A_2 in Eq. (3.35) depends on the implementation of Fast Fourier Transforms [see equation (3.34)]. We are using the FFTW implementation [22], which provides system dependent measurements (benchmarks) for a few choices of the gridsize n_f^3. To have better sampling we made our own measurements of FFT speed presented in Figure 3-11. On the vertical axis we plot the measurement of FFTW speed S_{FFT} in Megaflops, which, as expressed in terms of the workload coefficient $A_{\mathrm{FFT}}(c)$ measured in microseconds, will be $S_{\mathrm{FFT}}(c) = 2.5/(A_{\mathrm{FFT}}(c)\ln 2)$. Clearly, from the figure we observe that one should be very careful in choosing the fine density gridsize n_f: as the figure shows, the FFTW speed may differ by orders of magnitude even for close n_f.

Each choice for n_f used for mesh refinement carries memory and computational costs due to generating the fine mesh Green function. The machine memory constraint gives us the upper limit on the size of fine density mesh for the refinement.

Although $n_f = 32$ has a very high measurement of the FFT speed S_{FFT}, using the adaptive mesh refinement with this number in practice has shown not to result in any speedup in the overall measured workload w_{APP} over the non-adaptive workload w_{PP}. Instead, $n_f = 48$ appears to be the minimal gridsize that gives an advantage. Based

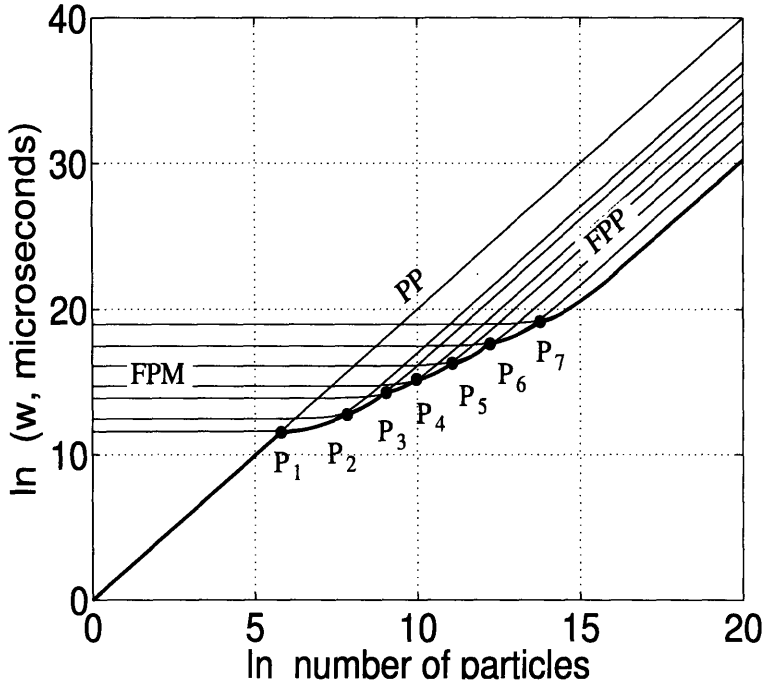

Figure 3-12: Model relations for cell workloads for a short range computation for the seven choices of the mesh refinement number, given by [equations (3.36) and (3.41)]. The points P_1–P_7 follow the linear slope given by Eq. (3.38). If the model is correct, depending on the number of particles within the cell a correct mesh refinement number can be chosen so that the workload for computing the short range forces is minimized.

on Figure 3-11 we arrive to a few practical choices for mesh refinement grid size

$$n_{\rm f}^i \equiv \{n_{\rm f}^1 \ldots n_{\rm f}^{n_{\rm max}}\} = \{48, 64, 96, 128, 192, 288, 448\} \tag{3.43}$$

to be used for mesh refinement. The mesh refinement grid sizes $n_{\rm f}^i$ given by equation (3.43) are roughly equally spaced in the logarithmic space and have high measurements for the speed of the FFT transform. The superscript index i or the *mesh refinement number* ranges within the interval $[1 \ldots n_{\rm max}]$, where $n_{\rm max}$ is the maximum mesh refinement number used for the adaptive force calculation [$n_{\rm max} = 7$ in the case of the set (3.43)]. The number of choices is limited since each additional choice of the mesh refinement gridsize results in the allocation of additional memory (see the discussion on memory requirements in Section 3.4). The speed values in Figure 3-11 are approximately equal to each other for the gridsizes given by (3.43), justifying our ignoging of the dependence of the coefficient A_2 in equation (3.37) on c.

If the values of the coefficients in Eqs. (3.33) and (3.35) are known, for each of the gridsizes of equation (3.43), one can construct the workload dependencies on the number of particles such as those shown in Figure 3-12 and choose the mesh refinement number that will minimize the PP-force computation workload based solely on the number of particles in the coarse mesh cell.

In practice one does not have the precise values for most coefficients in Eqs. (3.33) and (3.35). Moreover, these workload equations rest on the assumption of uniform particle distribution within the fine mesh simulation volume. From the discussion in Section 3.3.2 we know that the total workload is minimized when it is distributed equally between FPM and FPP for each cell

$$w_{\mathrm{FPM}} \approx w_{\mathrm{FPP}} \,. \tag{3.44}$$

Since we believe that this conclusion is not model dependent, this argument leads us to the introduction of the following total workload minimization scheme which is based solely on the direct cell workload measurements w_{FPM} and w_{FPP}.

Let us suppose a given coarse mesh cell is at the mesh refinement number i (somewhere within the curve segment P_iP_{i+1} in Figure 3-12) at a given timestep. If the cell is in the middle of the curve P_iP_{i+1}, its PP-force workload is evenly divided between FPP and FPM $w^i_{\mathrm{FPM}} = w^i_{\mathrm{FPP}}$. Now, if the number of particles within the cell has grown during the current timestep, the cell will move along the curve in the positive direction of the x-axis, while the share of FPP-direct summation $w^i_{\mathrm{FPP}}/w^i_{\mathrm{FPM}}$ is increasing. When the cell reaches the point P_{i+1} on the curve it changes to the next refinement number. The total workload continuity condition at the transition point reads

$$w^{i,\mathrm{up}}_{\mathrm{FPP}} + w^{i,\mathrm{up}}_{\mathrm{FPM}} = w^{i+1,\mathrm{down}}_{\mathrm{FPP}} + w^{i+1,\mathrm{down}}_{\mathrm{FPM}} \,, \tag{3.45}$$

where $w^{i,\mathrm{up}}_{\mathrm{FPP}}$, $w^{i,\mathrm{up}}_{\mathrm{FPM}}$ are the cell workloads measured in the simulation using mesh refinement number i

In our scheme, the change to the upper and lower mesh refinement numbers occurs when $w_{\mathrm{FPP}} > w^{i,\mathrm{up}}_{\mathrm{FPP}}$ and $w_{\mathrm{FPP}} < w^{i,\mathrm{down}}_{\mathrm{FPP}}$ respectively, where

$$w^{i,\mathrm{up}}_{\mathrm{FPP}} = \alpha^i w^i_{\mathrm{FPM}} \quad w^{i,\mathrm{down}}_{\mathrm{FPP}} = \beta^i w^i_{\mathrm{FPM}} \tag{3.46}$$

respectively, where α^i and β^i are the scheme parameters and the workloads w are directly measurable by timers. Setting $\beta^{i+1} \equiv 1/\alpha^i$ and using the continuity equation (3.45) we eliminate the w_{FPP} terms and obtain

$$\alpha^i = \frac{w^{i+1}_{\mathrm{FPM}}}{w^i_{\mathrm{FPM}}} \approx \frac{A_{\mathrm{FFT}}(c^{i+1})}{A_{\mathrm{FFT}}(c^i)} \left[\frac{c^{i+1}\log c^{i+1}}{c^i \log c^i}\right], \quad i = 1 \text{ to } 6 \,, \tag{3.47}$$

where $c^i \equiv (n^i_{\mathrm{f}})^3$. Since all the terms in the right-hand side are known or easy to measure we have the complete practical scheme of switching between the different mesh refinement numbers.

The curves in Figure 3-12 are constructed using the workload model coefficients measured in a real simulation. We observe that quasi-linear dependence 3.40 of the workload on the number of particles, similar to the one given by the envelope curve equation Eq. (3.38) is followed from $x \approx \exp(7)$ up to the extremely dense state of matter with $x = \exp(15)$ particles in the mesh refined coarse mesh cell, which is equivalent to $x/\tilde{R}^3_{\mathrm{c\,max}} = 1.5 \times 10^5$ overdensity above the average. Using the $i = 7$ mesh refinement number results in a factor of $\exp(10) \approx 2.2 \times 10^5$ speed increase over

the pure PP-direct summation, while using $i = 1$ instead would have resulted in only a factor $\exp(3) \approx 20$ speed advantage. At the other end of the curve $x = \exp(7)$, using the mesh refinements gridsize lower than $n_{\rm f}^1 = 48$ would show the timing that is more inferior than the timing of the pure PP-direct summation.

There is no scheme that can provide the continuity of the workload at the transition between the pure PP and the first mesh refinement number (the point P_1) without the direct measurements of both, since the former uses only 13 surrounding cells while the latter includes all the 26 surrounding cells within the fine mesh simulation volume (in order to take the full advantage of the FFTs). We apply *ad hoc* the following scheme

$$w_{\rm PP}^{\rm up} = \gamma^+ w_{\rm FPM}^1, \quad w_{\rm FPP}^{1,\rm down} = \gamma^- w_{\rm FPM}^1 \tag{3.48}$$

where γ_+ and γ_- are coefficients of the order of unity.

3.3.4 FPP particle data layout optimization

In Section 2.5 of Chapter 2 we introduced particle data layout optimization that results in a significant speedup for algorithms requiring frequent particle access based on their location in a chaining mesh cell (or HC mesh cell), by placing the data belonging to the same cell sequentially within the segments of the particle array. Once this *primary sorting* procedure is completed the speedup is achieved in PP-force calculation, because the particle data belonging to the same chaining mesh cell are placed sequentially within the segments of the particle array so that the CPU cache memory is used more effectively during the PP-force computation (see Section 2.6.3 of Chapter 2 and the references there for more discussion).

Even in the regime of heavy clustering a significant fraction of coarse chaining mesh cells are not dense enough and the short range forces for their particles are computed by the direct PP-summation. For the other cells the mesh refinement is performed and the short range forces are computed adaptively as described in Section 3.2. In the adaptive force computation the particle data access is performed most frequently during the FPP force calculation, and the particles are accessed based on their location in a fine chaining mesh cell defined in Section 3.2.1 and the number of particle accesses for a complete force computation scales as the square of the particle number within those cells. Putting the particle data belonging to the same fine chaining mesh within the segments of the particle array by performing an additional *secondary sorting* will reduce the time for computing the FPP forces.

As we know from Section 3.2.1, the complete fine mesh simulation volume for a coarse mesh cell for which the adaptive force computation is performed includes all the immediately adjacent coarse mesh cells, they all forming a subvolume of the simulation box of size $27(\Delta\tilde{x}_{\rm c})^3$. Due to the primary sorting procedure, the whole particle data within the fine mesh simulation volume occupies 27 segments of local or boundary layer particle arrays. In order to simplify the secondary sorting procedure

we impose a constraint on the fine chaining mesh size

$$\mathrm{mod}\ (n_{\mathrm{fc}}, 3) = 0\ , \tag{3.49}$$

making it possible to perform the secondary sorting procedure *in-place* within the 27 segments of particle arrays. Since original expression (3.6) does not in general satisfy this constraint, we use instead equation (3.7), which provides the maximum possible fine chaining mesh size satisfying constraint (3.49).

Upon the completion of the FPP force calculation, the secondary sorting must be undone for those of the 26 adjacent coarse mesh cells that belong to the non-local processes so that the computed accelerations are updated on those processes in the correct sequence.

In order to perform the secondary sorting for a segment of particles within the particle array we apply a similar sorting procedure using an extended linked list, as described in Section 2.5 of Chapter 2. Because the particle array members `g0` and `g1` are filled with the PM acceleration values and can not be used as pointers now, we allocate an auxiliary array in order to hold the forward and backward pointers of the extended linked list. In addition, each member of the auxiliary array holds the initial position of the corresponding particle within the particle array segment. This array member is used at the end of the FPP-force computation in order to undo the sorting. Once this sorting procedure is completed, we initialize the three dimensional array of pointers pointing to the segments of particles belonging to the same fine chaining mesh cell. When this array is used, the particles used for FPP-force computation are accessed sequentially in the array and we have achieved the optimal particle data layout for cache memory access.

The cost of performing the secondary sorting procedure for each mesh refinement is proportional to the number of particles in the fine mesh simulation volume, this cost is included into the timing for the FPP cell specific workload measurement.

3.4 Tests

In Chapter 2 we introduced scalable and load balanced P^3M code `llp3m-hc` and tested its performance. In this Chapter we introduced a new code `llap3m-hc` that takes advantage of all the techniques developed for `llap3m-hc`, but in addition uses a new method for adaptive PP-force calculation. In the following we analyze the performance of the new code and analyze the improvement over non-adaptive P^3M.

The new parameters introduced in equation (3.48) were set to values

$$\gamma^+ = 1.2, \quad \gamma^- = 0.5\ . \tag{3.50}$$

We did not perform extensive tests proving that these values are optimal, however qualitatively, this appears roughly to be the case. Using $i = 1$ for mesh refinement is effective when $w^1_{\mathrm{FPM}} \approx w^1_{\mathrm{FPP}}$, or $w^1_{\mathrm{APP}} = w^1_{\mathrm{FPM}} + w^1_{\mathrm{FPP}} \approx 2w^1_{\mathrm{FPM}}$. On the other hand, the adaptive mesh refinement completes roughly twice as much progress towards the completion of the PP-force computation as a whole, because as is seen

i	$n_{\rm f}$	Mflops	α^i	$w^i_{\rm FPM}$, sec
1	48	854.9	2.4	0.054
2	64	922.4	4.2	0.12
3	96	818.2	2.3	0.53
4	128	890.5	4.0	1.2
5	192	811.0	3.9	4.9
6	288	758.7	4.5	19
7	448	691.4	–	85

Table 3.2: Parameters specific for each mesh refinement number.

from equations (3.1) and (3.3) the adaptive mesh refinement performed on one cell updates forces on roughly twice as many pairs of particles as the PP-force computation. Therefore, the transition point between the pure PP and the lowest number in adaptive mesh refinement should occur at $w^1_{\rm APP} \approx 2w_{\rm PP}$ or $w_{\rm PP} \approx w^1_{\rm FPM}$, implying $\gamma^+ = 1$. On the other hand, if the first mesh refinement number is already used for a cell, and its $w^1_{\rm FPP} = \gamma^- w^1_{\rm FPM}$, then $w^1_{\rm APP} = (1+\gamma^-)w^1_{\rm FPM}$, implying $\gamma^- = 1$ at the transition point. Since the arguments above are very rough, setting $\gamma^+ = \gamma^- = 1$ may provide too shallow transition thresholds, rendering too frequent transitions between the PP and the lowest fine mesh gridsize APP. The parameter values in equation (3.50) provide higher threshold in both directions.

The parameter $C_{\rm f}$ was set to 15. In Table 3.2 we show the choices for the fine density mesh size for each mesh refinement number i. Also shown there is the FFTW speed measured in Megaflops and the estimate of the FPM workload for each mesh refinement number, which is obtained by multiplication of the number (10) of FFTs performed per each FPM force calculation by the wall clock time to make one FFT of the given size

$$w^i_{\rm FPM} \approx 10 \times \frac{2.5\, n_{\rm f}^3 \, \log_2(n_{\rm f}^3)}{\rm Mflops},\ 10^{-6}\ {\rm sec}\,. \tag{3.51}$$

The Hilbert Curve mesh is set equivalently to the coarse chaining mesh in our P^{3}M codes. We will therefore freely refer to the HC cells and the HC mesh when we discuss PP-force computation for which coarse chaining mesh is used.

In addition to the mesh refinements, the HC mesh is set differently in the adaptive code and non-blocking communications are implemented for parallel PM-force computation (see Sections 3.2.4 and 3.3.1).

The memory requirements for the `llap3m-hc` code are listed in Table 3.3.

Adaptive P^{3}M Simulation of ΛCDM

In Section 2.7.4 of Chapter 2 we presented the new `llap3m-hc` code and the test 800^3 particle simulation for the for the ΛCDM universe (Run 2 of Chapter 2, will be called Run A hereafter), ran on a 20 node cluster on 80 processes using Intel hyperthreading. The simulation evolved up to timestep 569 to the expansion factor $a = 0.68$ when it had to be stopped because at this point it took about two hours to advance the

Memory requirement	Notation	Memory Size, per process i	Total Memory size
Particle array	$M_{\rm P}$	4 bytes $\times$ $11N(i)$	4 bytes $\times$ $11N$
Particle linked list	$M_{\rm L}$	4 bytes $\times$ $N(i)$	4 bytes $\times$ N
HC mesh	$M_{\rm HC}$	23 bytes $\times$ $(1+\text{S.V.R.}^i)r_{\rm n}^i$	23 bytes $\times$ $n_h^0 n_h^1 n_h^2$
Table of HC entries	$M_{\rm K}$	4 bytes $\times$ $12 \times K \times n_{\rm pr}$	4 bytes $\times$ $12 \times K \times n_{\rm pr}^2$
Green function	$M_{\rm G}$	4 bytes $\times$ $\texttt{nlta}(i)\, n^1(n^2/2+1)$	4 bytes $\times$ $n^0 n^1(n^2/2+1)$
PM density and force meshes	$M_{\rm PM}$	4 bytes $\times$ $2\,\texttt{nlta}(i)\, n^1(n^2+2)$	4 bytes $\times$ $2n^0 n^1(n^2+2)$
PM non-blocking communications	$M_{\rm NBPM}$	flexible	flexible
Optional FFTW scratch space	$M_{\rm FFT}$	4 bytes $\times$ $\texttt{nlta}(i)\, n^1(n^2/2+1)$	4 bytes $\times$ $n^0 n^1(n^2/2+1)$
PP boundary layer particles	$M_{\rm PP}$	4 bytes $\times$ $8\Delta n_{\rm PP}$	4 bytes $\times$ $8\sum \Delta n_{\rm PP}$
Fine mesh Green function	$M_{\rm GF}$	4 bytes $\times$ $\sum_i (n_{\rm f}^i/2+1)^3$	See the caption

Table 3.3: Dominant memory requirements of parallel `l1ap3m-hc` code. By definition S.V.R.^{i} is the ratio of the number of the boundary layer cells to the number of local cells on process i. The fine mesh Green function memory requirement is given per mesh refinement number i used. The summation in the fine mesh green function memory requirement is over the refinement numbers acutally used by the process for the adaptive mesh refinement of its local cells in the last few timesteps.

problem by one timestep. The simulation was completed in 13.3 days wall clock time and 10.7 days CPU time respectively (time is cumulative and averaged over all the processes of the cluster). At the end of the simulation, the force computation time was dominated by the PP-direct summations.

We performed a new run (here *Run 1*), identical to Run A, using the same number of processes and nodes as before, now with our new adaptive P^{3}M `l1ap3m-hc` code.

The Layzer-Irvine energy conservation check [equation (2.7)] was satisfied to a precision 4×10^{-5} at timestep 569 (compare 4×10^{-5} for run A) and to $\tilde{E}_{\rm con}/\tilde{E}_g = -5 \times 10^{-5}$ at the end of the run.

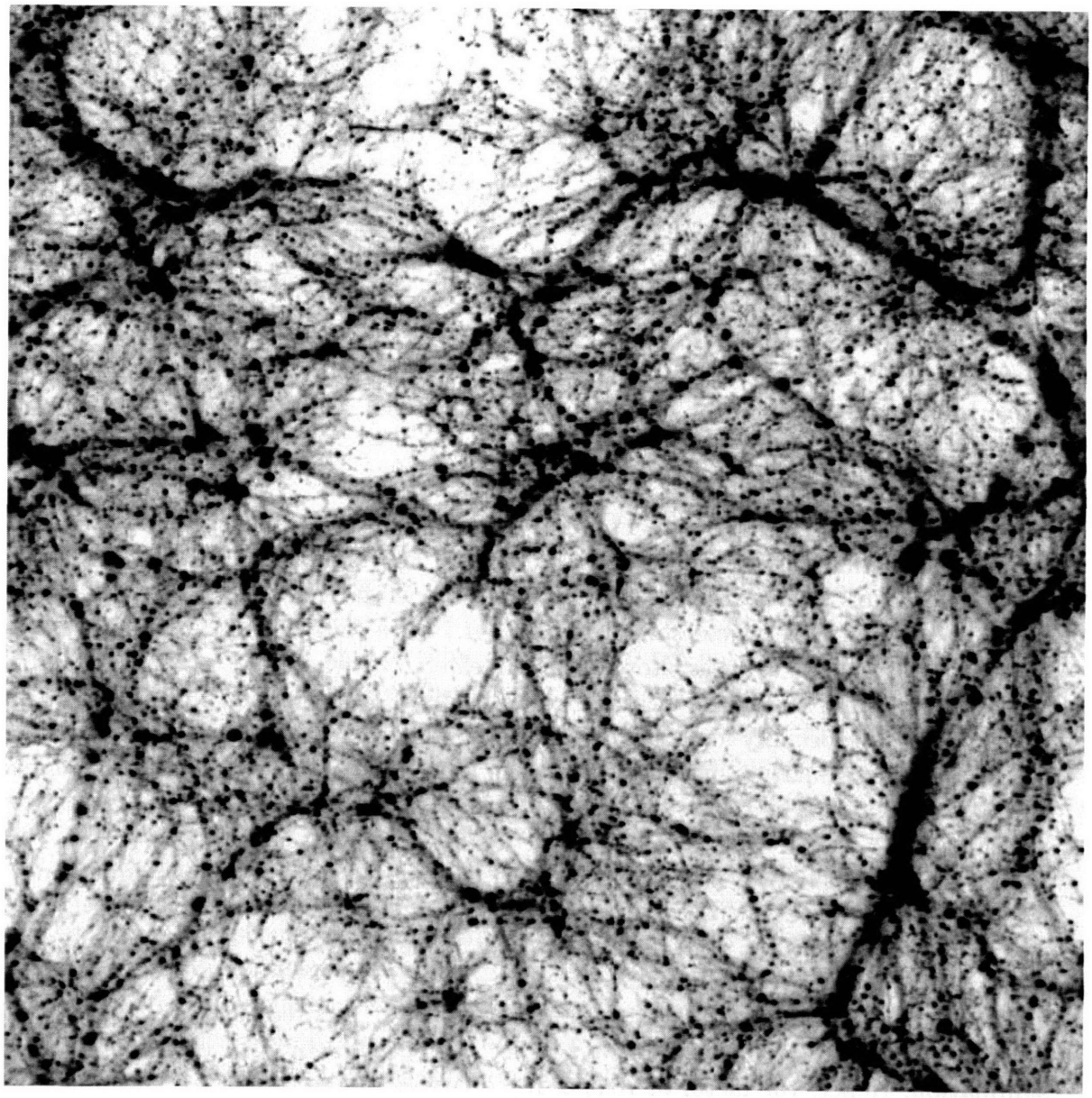

Figure 3-13: Matter density distribution shown at $z = 0$, at the last timestep of the adaptive ΛCDM run described in Section 3.4.

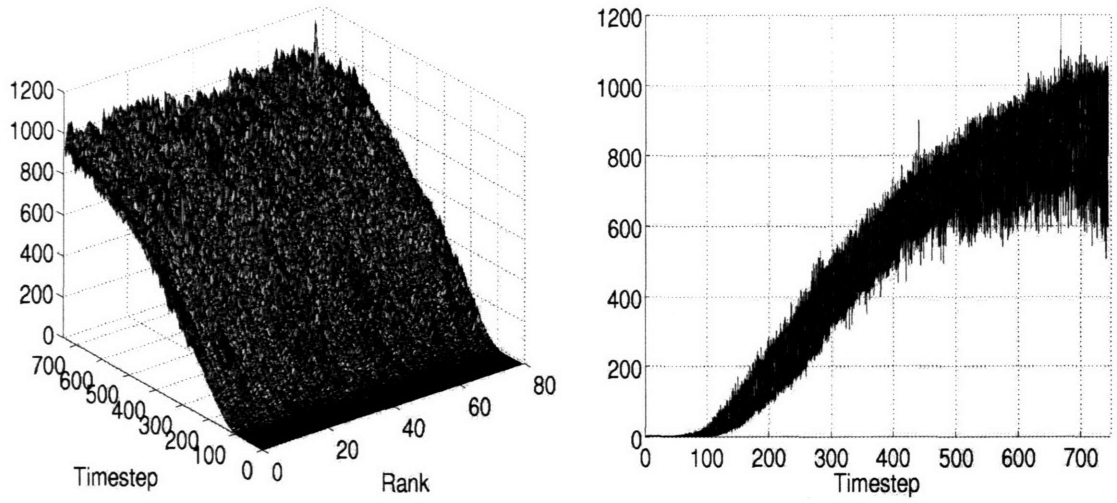

Figure 3-14: Number of local cells for which mesh refinement was performed for short range force computation. Left: view from oblique angle. Right: projected view down the Rank (process number) axis.

As expected, after we implemented the adaptive PP-force calculation we observed great improvement in the timing performance. The simulation reached the final Timestep 569 of Run A in just 3.0 days (speedup of 4.4), and 2.0 days CPU time (speedup of 5.3). The whole simulation took 4.7 days wall clock time and 3.3 days CPU time to evolve. It was stopped at timestep 743 where the expansion factor reached $a = 1$, and the image of matter density distribution is shown in Figure 3-13. The speedup is especially high if we compare the measurements of the wall clock time to advance the simulation one timestep at the last timestep 569 of the old run, which are respectively 7400 seconds in Run A and just 743 seconds in Run 1, showing factor of 10 speedup.

Let us notice that the spacing of the HC mesh cells is set up slightly differently from those in Run A where we used $\tilde{R}_{\mathrm{c\,max}} = 2.78$. Due to the new relation (3.26) based on force accuracy considerations, $\tilde{R}_{\mathrm{c\,max}}$ was now set to 2.3 leading to the increase in the size of the HC mesh now being $347^3 \approx 4.2 \times 10^7$ cells. Since the total PP-direct summation workload is proportional to $\tilde{R}^3_{\mathrm{c\,max}}$ [c.f. equation (3.33)], on the basis of the reduced $\tilde{R}_{\mathrm{c\,max}}$ alone we predict the speedup by 76% of the PP-direct summation. Hence, the factor of 10 speedup observed at the timestep 569 can not be attributed to reducing the HC mesh spacing alone.

The key difference responsible for this order of magnitude speedup is the adaptive PP-force computation. In Figure 3-14 we present the number of local HC cells for which the PP-forces were computed adaptively as a function of process and the timestep. At the beginning of the run the total number of mesh refined cells in the whole simulation box is less than a dozen, since the clustering of matter in the initial conditions is low. As the clustering grows, some cells become very dense and the number of mesh refined cells steadily grows as it is no longer efficient to compute

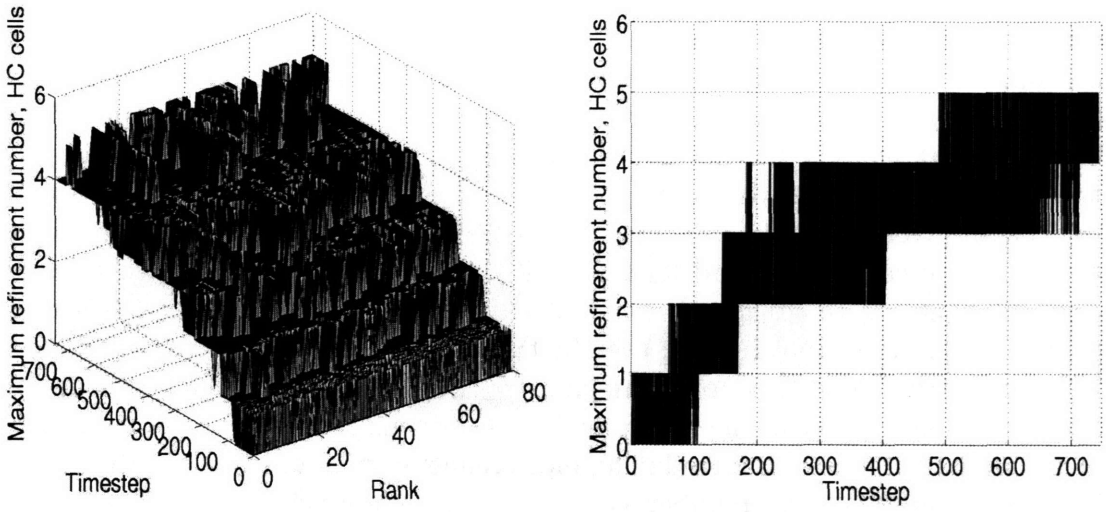

Figure 3-15: Maximum refinement number for a local HC cell as a function of Rank and the Timestep.

the PP-forces for those cells by pure direct summation [c.f equation (3.42)]. By the timestep 600 there are on average 840 mesh refined cells on each process (0.16% of the total average number of local cells). At this point the number appears to reach saturation, happening when most of the matter has clustered and no new clusters are forming in between the already existing ones. It is amazing that by the refinement of just 0.16% cells, we were able to achieve a factor of ten speedup at timestep 569.

As the clusters of particles evolve and become more dense, the number density of particles within the cells they occupy grows, leading to higher FPP-load computed by the direct summation. When the load rises highly enough to reach the critical value $w_{\mathrm{FPP}}^{i,\mathrm{up}}$, given by equation (3.46) where i is the current refinement number of the cell under consideration, the refinement number of the cell is switched to $i+1$ to keep the growth of the workload of the cell with the number of particles close to linear [see equation (3.40)]. If the cluster of matter moves elsewhere, the w_{FPP} workload of the cell it had occupied before reduces and once it has fallen below $w_{\mathrm{FPP}}^{i,\mathrm{down}}$, the refinement number of the cell switches to $i-1$, while the refinement number of a new cell occupied by the cluster switches to the same $i+1$.

The above description gives us some sense on how the refinement numbers are distributed in cells within the simulation volume. Even if the clusters are moving, the maximum refinement number of the cells in the central region of their halo steadily grows. In Figure 3-15 we present the maximum refinement number of local HC cells as a function of process and the timestep. From figure we observe the characteristic rise of the maximum refinement number as the time evolves. In the beginning of the run, the lowest possible refinement number was used for all of those very few HC mesh cells that were refined. At the end of the run, the maximum refinement number of a cell within the whole simulation volume reached 5, meaning that maximum size of fine density mesh used for mesh refinement was 192 (see Table 3.2).

If the simulation were evolved to the later stages of clustering, beyond timestep 743, we would expect the number of mesh refined cells to shrink as the existing clusters of dark matter merge, and the maximum refinement number per process to rise as the clusters become more dense. Finally, either they arrive to dynamical equilibrium or the refinement number reaches the maximum (7) allowed by the current implementation, which corresponds to the maximum resolution fine density grid available $n_f = 448$ (given by Table 3.2). Once the latter limiting case is reached, no finer FPM density grid can be used and the FPP cell workload starts to grow quadratically (as given by equation (3.36) at $n_f = 448$) if number of particles it contains further increases. This limit was not reached even by the densest of the cells in our simulation run. The quadratic growth is not a fundamental problem for our adaptive code. If the current dynamic range of the available fine mesh grids proves to be not sufficient, it can simply be extended by introducing more refinement numbers into Table 3.2.

The maximum refinement number available for the adaptive scheme is determined by the machine memory constraint. The amount of memory required to store a fine mesh Green function of gridsize n_f is 4 bytes $\times$ $(n_f/2 + 1)^3$. Since the values n_f^i, where $i \in [1 \ldots n_{max}]$, are chosen to sparsely and uniformly sample the space of $\log n_f$, the fine mesh Green function memory requirement M_{GF} is dominated by $n_f^{n_{max}}$. The largest gridsize used for mesh refinement is given by

$$n_f^{n_{max}} \approx 580 \times \left(\frac{M_{GF}}{100\,\mathrm{MB}} \right)^{1/3} . \tag{3.52}$$

It is noticeable in the figures discussed that at any timestep the data for the local domain of each process look roughly similar despite the very high clustering achieved in the simulation. At the very end of the run each local domain uses mesh refinement for approximately the same number of cells (840 ± 200), and the maximum refinement number for all the processes is either 4 or 5. This point is very important and we believe will certainly apply for a generic adaptive P^{3}M simulation. By contrast, in the identical Run A, a situation was easily reached when some processes contain only a few HC cells, while others contain hundreds of thousands of cells. For Run A this leads to a highly non-uniform distribution of memory among the nodes where special care was required to avoid overflow of memory. We can suggest now just by looking at Figure 3-14 that the total problem memory load, which is dominated by particle data, is better distributed in our Run 1 than in Run A. This suggestion will be tested further in this section.

In Figures 3-16 (compare with Figure 2-18) and 3-17 we plot the local domain sizes of the processes and their particle content as a function of rank and the timestep during the whole simulation. The dynamic range of the number of particles and the volume domains taken among all the processes and all the timesteps in the simulation shows a striking difference with the same numbers for Run A. Indeed, during Run A the number of cells within the local domain ranged from 1 to 1.1×10^6 (being on average 3.0×10^5) while the number local particles ranged from 4.7×10^4 to 1.8×10^7. In Run 1, the number of local cells ranges from 2.4×10^4 to 1.6×10^6 (being on average 5.2×10^5), and the number of local particles ranges from 3×10^6 to 1.1×10^7.

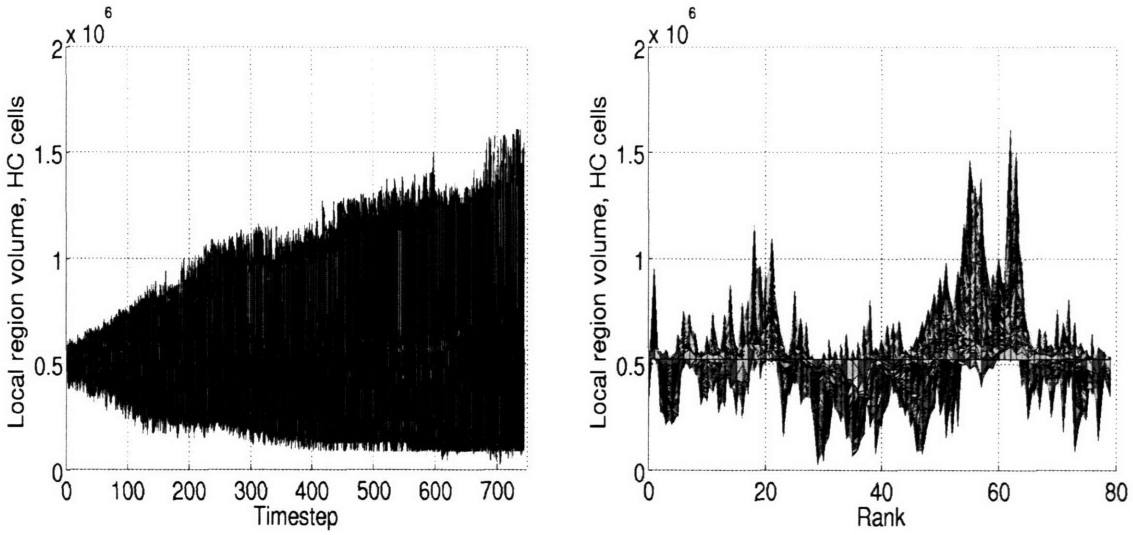

Figure 3-16: Volume of HC local regions, as a function of timestep and process rank.

In comparison with Run A, the dynamic ranges of the variations of the local number of cells and local particles reduced by orders of magnitude.

This is a rather striking and favorable change in the dynamic range of these variables. It is favorable because it provides a more uniform and therefore more efficient memory distribution among the processes as we already suggested. Precisely this change is caused by the fact that the CPU cell workload which is used for balancing in both our present code and Chapter 2 is a quadratic measure of the number of particles in a cell for the non-adaptive PP [equation (3.41)] and is linear measure for the adaptive-PP [equation (3.39)]. By balancing the CPU workloads in the adaptive-PP code we are managing at the same time to roughly balance the number of particles in the local regions being proportional to the number of particles. Catching both rabbits was not possible using the non-adaptive PP.

In Figure 3-18 (a) we present the cell statistics for two timesteps (283 and 743) in order to test the workload minimization scheme described in Section 3.3.3 and test how well it replicates model relations predicting in particular the quasi-linear dependence of cell adaptive-PP workload on the number of particles [equations (3.38), (3.39) and (3.41)]. Each point on the plot is a geometric average of the effective workloads of all the HC-cells in the simulation volume whose effective workload falls within a bin in the logarithmic space of the number of particles, 10^4 bins are sampling the range between 10 and 10^6 particles per bin. The horizontal line approximately divides the domain where mesh refinement is performed (the region above the line) from the domain where PP-direct summation takes place (below the line). The position on the vertical axis of the solid line is given by the workload $w \approx w^1_{\rm FPM}$ [c.f. equations (3.48) and (3.50)]. The plot of the workload against the number of particles shows a broken power law with index $w \propto x^{1.1}$ at $x \geq 3000$ and $w \propto x^{1.61}$ at $x < 730$. In addition, a small bump of relative magnitude of $\approx 20\%$ is present in the transition interval $730 \leq x < 3000$ for both timesteps.

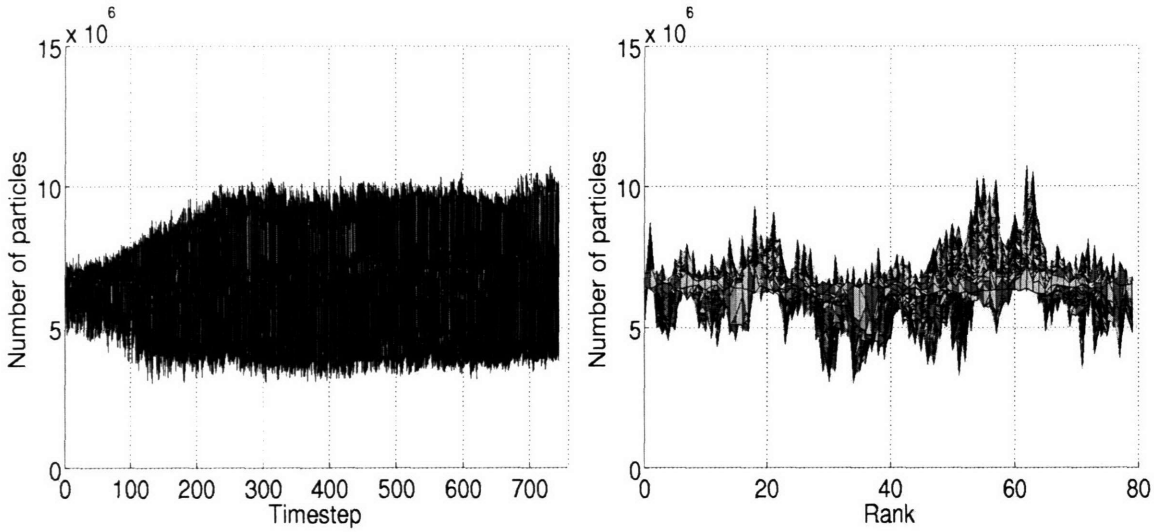

Figure 3-17: Local number of particles, as a function of timestep and process rank.

In the practical workload minimization scheme, described in Section 3.3.3 the particle number is not used for the decision on mesh refinement. Instead, only timing measurements are used. However, this scheme was designed to replicate the workload model described in Section 3.3.2, where the number of particles enters quasi-linearly in the relation (3.39) for the cell workload. The power law index 1.1 measured in the figure for high number of particles $x \geq 3000$ is in good agreement with the predicted quasi-linearity.

The high particle number cells on the plot can be divided into segments corresponding to certain mesh refinement by drawing horizontal lines at the positions $w^{i,\mathrm{up}}_{\mathrm{APP}} = w^i_{\mathrm{FPP}} + w^i_{\mathrm{FPM}} = (\alpha^i + 1) w^i_{\mathrm{FPM}}$, given by the continuity equation (3.46). By comparing the cell positions on the plot at the two timesteps (283 and 743), we observe that as the simulation evolves, cells move to the right along the curve on the figure while their refinement number increases. Indeed, in Figure 3-18 (b) we observe that the fraction of cells containing many particles increases with time and in Figure 3-18 (c) we observe that the fraction of time spent doing adaptive-PP force computation for those cells increases. From the data plotted in Figures 3-18 (b) and (c) we find that at timestep 743, as much as 4% of the total force evaluation workload is spent doing force computation for just 0.04% of the total number of cells in the simulation volume. Those cells all have relatively high workload since each of them contains more than 3.9×10^4 particles.

The change in the power law slope occurs when the adaptive PP-force calculation turns on, according to equation (3.48) for the cells that take too much time for their PP-force calculation by direct summation. The power law index 1.61 measured for low workload cells appears to be in disagreement with our model of quadratic dependence (3.41) for the workload of cells whose PP-forces are computed by direct summation. This disagreement is likely due to the inaccuracy of the uniform matter distribution approximation used to derive (3.41) on the scales of the coarse chaining

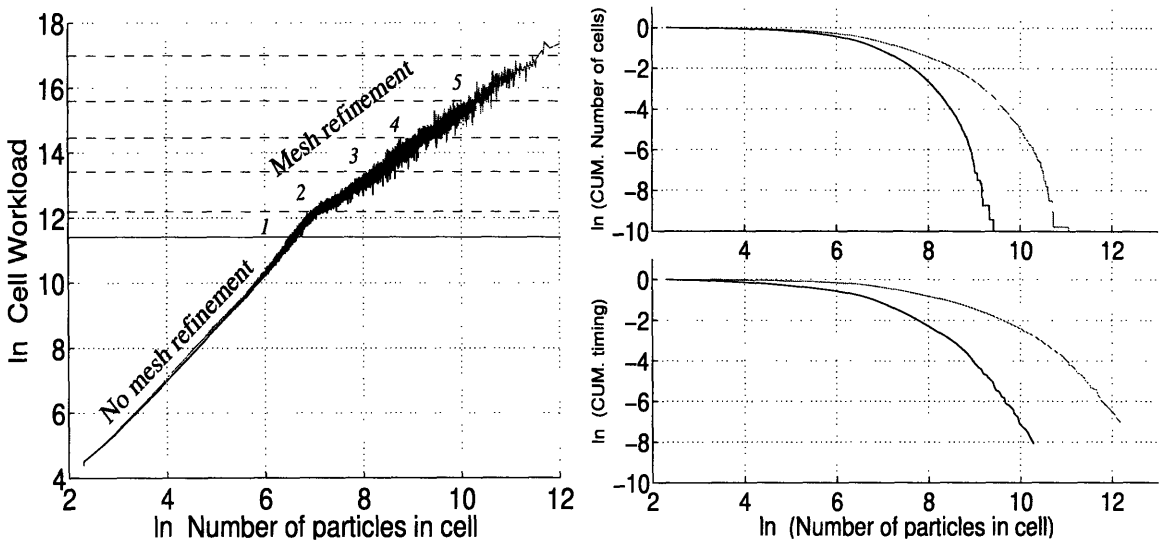

Figure 3-18: Cell statistics at timestep 283 of Run 1 (black curve) and at the last timestep 743 (red curve). Left (a): The cell workload at timesteps 283 (black curve) and 743 (red curve). The dashed lines split the cells into the segments corresponding to their mesh refinement numbers. Right, upper panel (b): natural logarithm of the fraction of the number of cells with higher number of particles than given by the horizontal axis (CUM.=cumulative). Right, lower panel (c): natural logarithm of the fraction of the workload in cells with higher number of particles than given by the horizontal axis.

mesh spacing. In addition, in the limit of low particle numbers within the cell, the PM-fraction of the total workload, scaling quasi-linearly with the number of particles, becomes significant, changing the power law index at low particle number. In Figure 3-18 (a) we observe a slight decrease in the power law index of the workload dependence at small particle numbers, within the range of 9 to 20 particles.

The small bump in the average workload for the cells in the transition region $730 \leq x < 3000$ is due to the cells whose workload satisfies *neither* of the two equations (3.48). Our practical scheme does not provide the optimal solution for these, and in practice some cells keep switching between computing their PP-forces adaptively and non-adaptively from one timestep to another until their clustering changes so that they move outside the transition region. This uncertainty results in a 20% relative magnitude increase in the average workload for these cells. These cells also cause fluctuations in the local workload measurements that are deleterious for the load balancing.

Let us now turn to analyzing the load balancing of the simulation. In Figure 3-19 we present the plot of the effective workload as a function of timestep and rank (process number). The most important difference from same plot for P^3M (Run A) is the great difference in the magnitude of the average load at late timesteps and their growth tendencies. These differences are mostly due to the mesh refinement performed for heavy workload cells. At timestep 569, the average workload has decreased by a factor of 13. Note that this number is different from the overall wall clock time

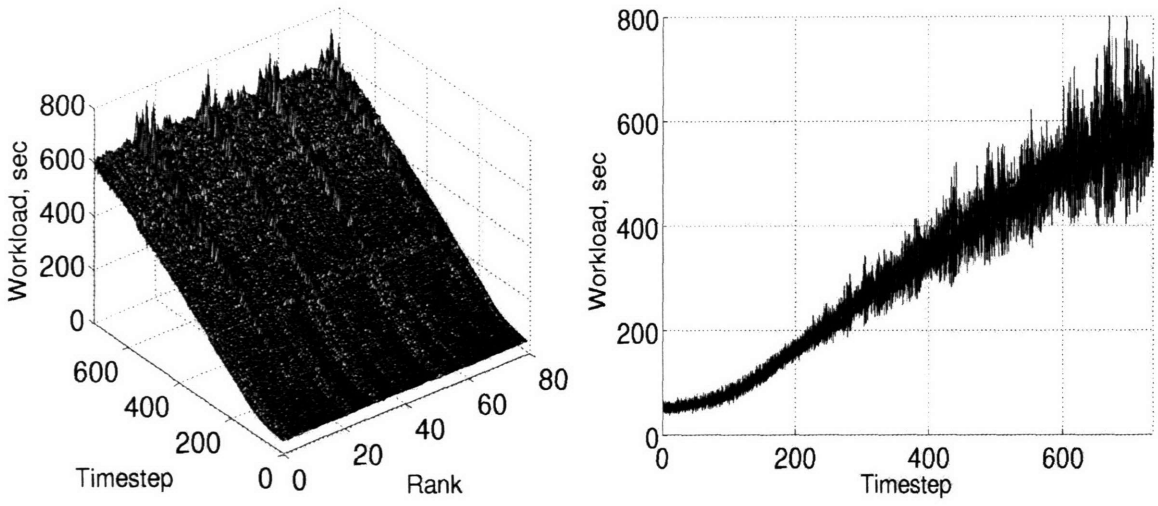

Figure 3-19: Effective workload of each process as a function of timestep. Views from two different directions.

per timestep speedup factor of 10, because the wall clock time per timestep is not a perfect measure of the process workload, it depends in addition to such factors, such as the value of load imbalance.

Another very important difference is that all deviations from the perfect load balance (when all ranks have the same workload) appear to be purely random in Figure 3-19. There is no systematics other that caused by the CPU fluctuations on a single node, leading to the random spikes across all the timesteps at four certain ranks. The systematic deviation from perfect load balance occurred in Run A in particular due to the processes whose workload is dominated by heavy workload cells. Those cells contained enough particles so that their cell loads were comparable with the average workload of a process, making it very hard or impossible to achieve perfect residual load imbalance due to the granularity of the workload. Although the systematic deviations appeared only slightly in the Run A the problem would definitely become vital if we evolved the simulation to the later time. In our present runs the distribution of matter at every timestep has the same clustering as in Run A. The cells occupied by those high density clumps of matter have so many particles that their short range forces are now computed adaptively using a high mesh refinement number. In the result, the cell workloads determining the maximum granularity of the total workload are much less than the average workload per process.

Indeed, from Figure 3-15 we find that at the end of the run as well as at timestep 590 the highest refinement number in the whole simulation box was $i = 5$, which gives us the maximum size of the fine mesh density grid $n_{\mathrm{f}} = 192$ used for FPM. We can approximate the maximum cell workload in the simulation by using the measurement of FFTW speed shown in Figure 3-11 for this gridsize (811.1 Mflops), since the FPM workload is dominated by the FFT cost and we know that there are currently $M_{\mathrm{FPM}} = 10$ FFTs performed each FPM force computation. Using the argument $w_{\mathrm{FPM}} \approx w_{\mathrm{FPP}}$ [equation (3.44)] of Section 3.3.3 we find that the cell workload of a cell

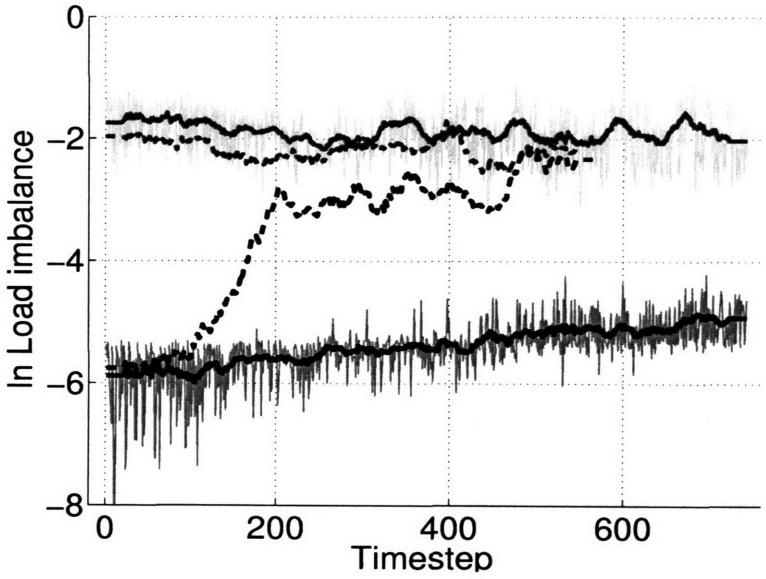

Figure 3-20: Instantaneous (blue and yellow solid curves), and residual (pink curve and black solid curves) load imbalance as a function of timestep for the 800^3 adaptive P^3M with Hilbert curve repartitioning on 80 processes. The dashed curves show the same data Run A.

refined with $i = 5$ is $w_{\text{APP}} = 10$ sec, which is indeed much less than the average workload (see Figure 3-19). Performing the same procedure for the maximum refinement number available $i = 7$ of those listed in equation (3.43), we obtain the maximum value of the cell workload performed in an optimal way (when $w_{\text{FPM}} \approx w_{\text{FPP}}$) by the adaptive mesh refinement scheme $w^7_{\text{APP}} \approx 170$ sec, which is still below the average workload per process, yielding low granularity.

Given the above arguments for the maximum cell workload, we now predict that due to the low granularity of the total workload, the repartitioning technique described in Chapter 2 must be very effective in finding the optimal target partitioning state leading to low residual load imbalance. In Figure 3-20 we present the measured instantaneous and residual load imbalance for each timestep in Run 1 and compare with the data for Run A. Repartitioning indeed is very effective in minimizing the load imbalance Run 1. Due to the low workload granularity, the residual load imbalance does not exceed 5×10^{-3} at any point during the run. In Run A the residual load imbalance grows setting the absolute lower limit for the instantaneous load imbalance. If the run were evolved to later timesteps the residual load imbalance would have grown even higher leading to a substantial instantaneous load imbalance.

The other pattern of importance visible in Figure 3-20 is a systematically higher average instantaneous load imbalance at the beginning of the run, where the effect of mesh refinement is negligible. One of the likely effects causing this systematic difference is the effect of the microsecond accuracy of the cell workload measurements by function `ntp_gettime`, which provides better relative accuracy for the cell measurements in Run A, where the average workload per cell is 252 μsec at the beginning of

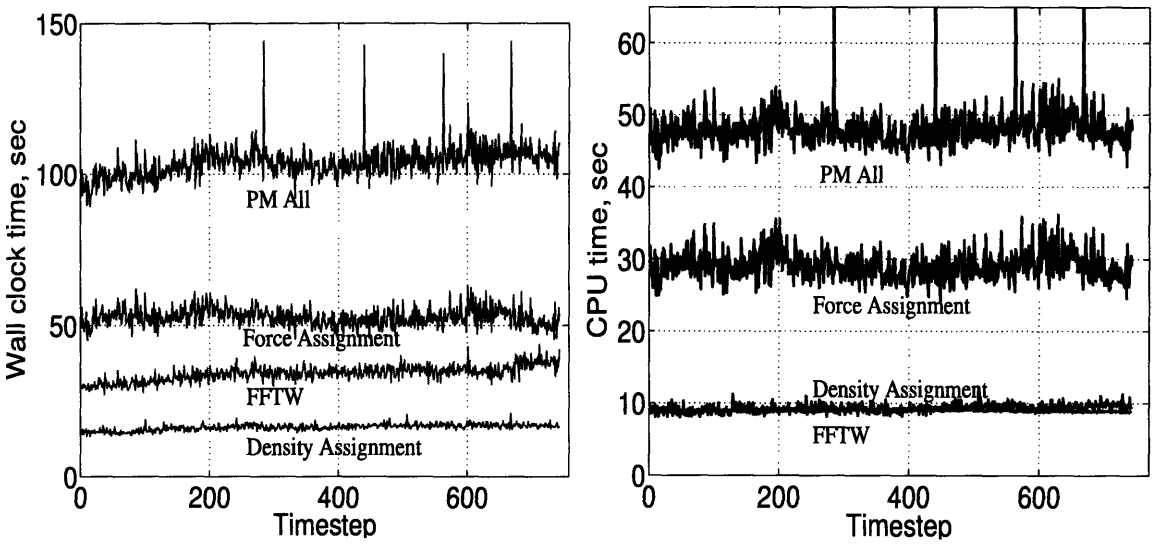

Figure 3-21: Structure of the wall clock time (Left panel) and the CPU time (right panel) per timestep of PM-force computation. Time is averaged per process. The four spikes are due to recomputation of the Green function when the code is restarted every 24 hours.

the run, as opposed to to 96 μsec for Run 1. Since the average number of cells per process has increased from 2.96×10^5 (for Run A) to 5.22×10^5 (for Run 1), this higher relative error was accumulated a larger number of times for Run 1, causing bigger errors in the measurements of the local workloads used for load balancing, leading to higher values of the measured instantaneous load imbalance.

Let us now turn to the analysis of the workload contributions of different components of force.

In Figure 3-21 we present the timing of different components of PM-force calculation, and compare it with Figure 2-19 for Run A. There are two differences of the PM between these two runs. First, non-blocking communications were introduced, as described in Section 3.3.1. Second, the local domains of the processes have the lower dynamic range during Run 1, as compared with the non-adaptive Run A (as shown in Figures 3-16 compared to Figure 2-18), influencing the exchange between the local domains and the FFTW slab domains during the PM-force computation. The differences between Figures 3-21 and 2-19 are due to both reasons. There are two improvements in the PM-force Wall clock time performance in Run 1 over Run A. First, the wall clock time per timestep in the PM-run has increased by $\approx 25\%$ towards the end of Run A, while remaining almost a constant in Run 1. Second, the average PM wall clock time is 50% less in Run 1 by as compared to Run A. Contrary to the improvement in the wall clock time performance, the CPU time performance worsens, showing a factor of 3 slowdown. The rise of the CPU time is attributed exclusively to the rise in $W_{\rm NL}$ and $W_{\rm wait}$ terms in equations (3.31) and (3.32), or the terms that can not be separated into the cell workloads.

The wall clock and CPU timing measurements for short range force computa-

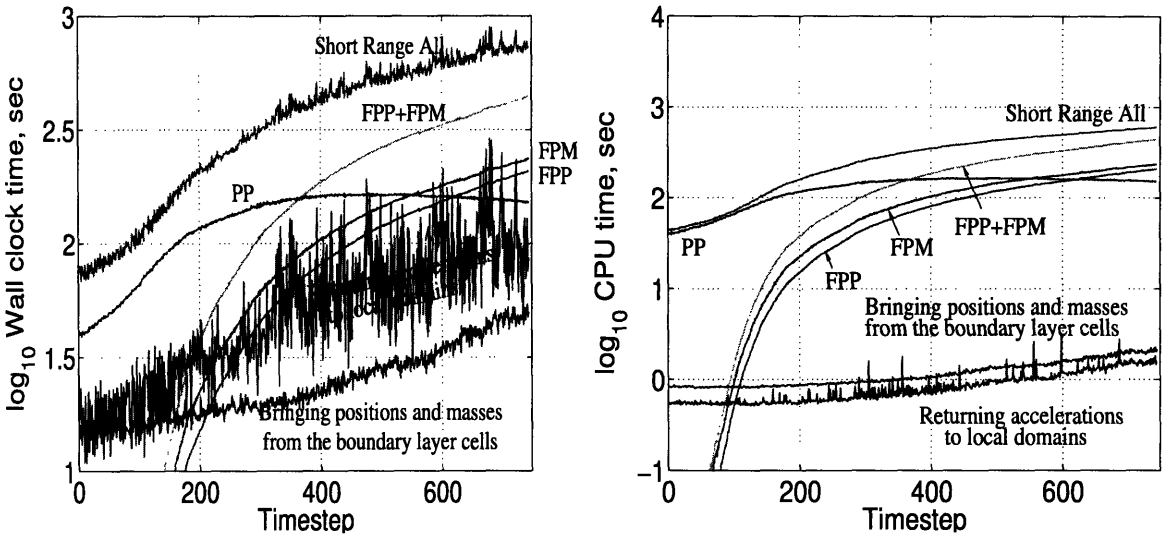

Figure 3-22: Structure of the wall clock time and CPU time per timestep of the PP-force computation. Time is averaged per process. Cell specific workloads are labeled as FPM, FPP and PP (PP-direct summation). Also plotted are times to bring the boundary layer particles from the other processes and bring back the computed accelerations. The uppermost curve shows the total workload for short range forces.

tion are presented in Figure 3-22. The whole PP-force computation workload is decomposed into the HC-cell specific portion $\langle W_{\rm HC}\rangle$ of the total CPU workload equation (3.32) and all the rest: $\langle W_{\rm wait}\rangle$ and $\langle W_{\rm NL}\rangle$. The timer measurements for computing the short range forces labeled by FPM, FPP and PP are parts of $\langle W_{\rm HC}\rangle$ since they only use the particle data contained within the short range force simulation volumes of each of their local HC-cells.

Each timestep, before the short range force computation takes place, the particle data of the local region boundary layer cells need to be brought from the other processes for computing the short range forces for the local particles (see Figure 2-8). The computed accelerations of the boundary layer cell particles are then brought back. These two steps involve interprocess communications and belong to the $\langle W_{\rm wait}\rangle$ portion of the workload. These contributions are plotted in Figure 3-22 and they are fully analogous to the data plotted in Figure 2-20.

In agreement with the expectations, we observe that at the beginning of the simulation, most of the total cell-specific workload is due to the PP-direct summation. As the workloads of the cells where clusters of matter are forming grow, adaptive mesh refinement is used for the short range force computations for their particles. The number of mesh refined cells quickly increases with clustering (c.f. Figure 3-14) and these cells become denser, taking particles away from the adjacent cells and reducing their load. At timestep 370 the PP-forces are computed adaptively for just 0.1% fraction of the total number of HC-cells, however their workload contribution into the total workload of short range force computation reaches 50%, as is seen from Figure 3-22.

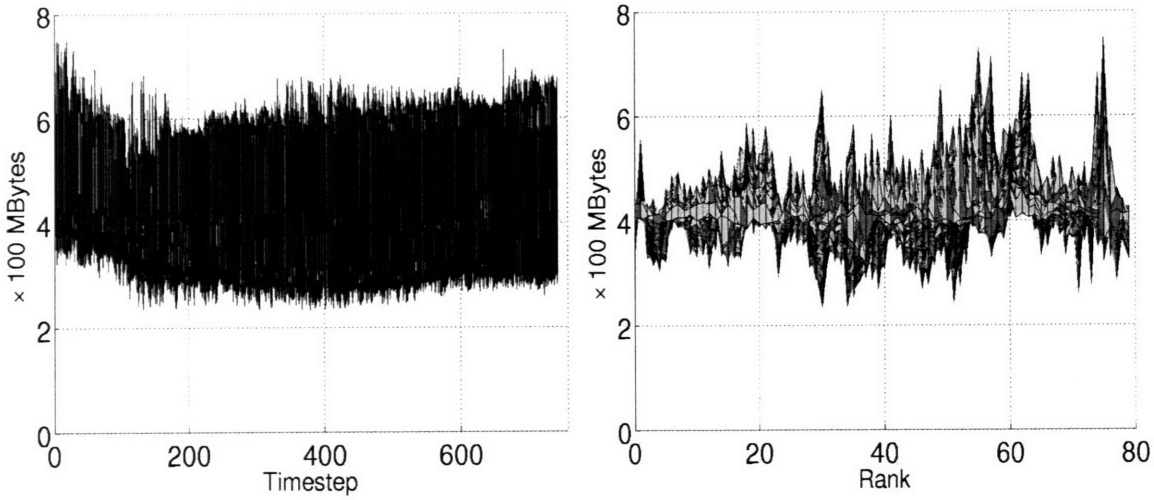

Figure 3-23: Maximum total memory allocated by any node as a function of timestep (left panel) and rank on each node (right panel).

Regarding our practical scheme for mesh refinement (Section 3.3.3) it is useful to note in Figure 3-22 that we have on average $\langle w_{\mathrm{FPP}} \rangle < \langle w_{\mathrm{FPM}} \rangle$. The ratio $\langle w_{\mathrm{FPM}} \rangle / \langle w_{\mathrm{FPP}} \rangle$ systematically decreases during the run, lowering from ≈ 1.36 at timestep 300, and ≈ 1.3 at timestep 370 to ≈ 1.15 at timestep 743. This observation shows that our workload selection scheme designed to divide the adaptive-PP workload amount equally between FPP and FPM works well, but not perfectly. However at high clustering, where the contribution of adaptive force computation in the workload is significant, the balance between FPP and FPM improves.

We now analyze the direct measurements of memory usage. In Figure 3-23 we present the maximum amount of memory allocated by each process as a function of rank of the process and timestep. The dynamic range of the plotted value is 10 for Run A and just 3.2 for Run 1. In agreement with the arguments for memory balancing presented above, we conclude that memory is balanced much better than in the non-adaptive $\mathrm{P^3M}$ run (see Figure 2-21). Averaged over the processes, the maximum required memory per timestep evolved from $\approx 410\,\mathrm{MB}$ at the beginning of the run to 430 MB at the end. The same data for Run A did not have tendency to increase, instead it fluctuates within the range 369 MB to 378 MB. We observe about 11% increase for memory requirement in the Run 1 over Run A.

One of the reasons for the memory increase is the higher size of the HC mesh caused by the reduced $\tilde{R}_{\mathrm{c\,max}} \approx 2.3$ as compared with $\tilde{R}_{\mathrm{c\,max}} \approx 2.78$ for Run A. However this alone can explain only a few percent of the difference. In order to understand the reason for increased memory requirement, in Figure 3-24 we present the memory usage on process 1 during the first two timesteps of Run 1. The dashed horizontal line shows the memory requirement based on the number of particles (6.44×10^6), the number of local HC-cells (5.70×10^5), and the number number of entries of Hilbert curve into the simulation volume $K = 1.02 \times 10^5$. Using Table 3.3, we find $M_{\mathrm{P}} + M_{\mathrm{K}} +$

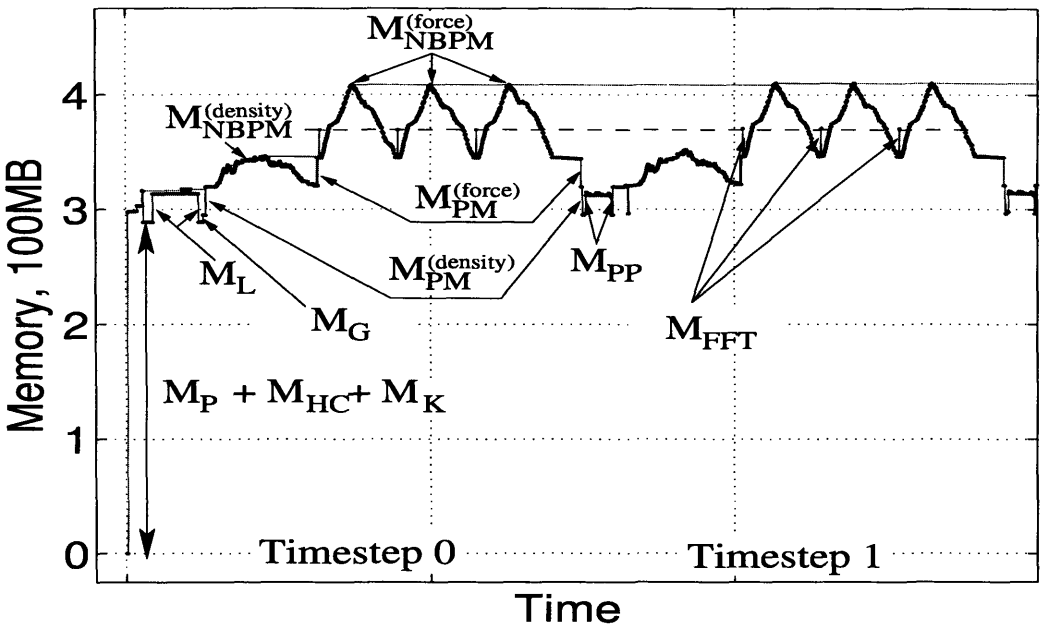

Figure 3-24: Itemized memory usage on process 0 during the first two timesteps of Run 1 (the periodically repeated pattern) in horizontal direction. All the memory requirements listed in Table 3.3 are labeled.

$M_{HC} + M_G + M_{PM} + M_{FFT} = 367\,\text{MB}$ (shown on the plot by the horizontal dashed line). The memory requirement grows above this limit due to the implementation of non-blocking PM communications, which allows the code to advance past the communication requests while accumulating the memory buffers needed in order to receive the incoming messages. The memory amount associated with non-blocking PM message requests M_{NBPM} is currently unconstrained, however and is the main reason why the memory requirement has grown for Run 1 in comparison with Run A. It is possible to constrain this memory amount explicitly once the optimal scheme for doing so is worked out.

3.4.1 Scalability test

We now analyze the scalability of our adaptive `llap3m-hc` code by running the same problem on different number of processes up to the final expansion factor $a = 1$ and measuring the total wall clock time it takes to evolve the problem. The analysis here is similar to the one presented in Section 2.7.5 where the identical simulations were presented with the only difference that the PP-forces were computed by direct summation (non-adaptive PP).

Only two choices for the particle number and density mesh are tested (288^3 and 384^3). By using smaller size problems than possible on such a large number of processes we should keep in mind that the scalability result will improve for larger problem size performed on the same nodes due to higher discretization of local do-

Run	$N_{\rm gr}$	Nodes	$n_{\rm pr}$	Refinements	Speedup	(W1)	(W2)	Scalability, S
3a	288^3	1	4	694,875	1.65	47.76h	5.43h	-1
3b	288^3	2	8	671,053	1.68	49.28h	8.08h	-0.96
3c	288^3	3	12	666,758	1.72	50.64h	9.54h	-0.95
3d	288^3	4	16	673,700	1.72	53.04h	10.50h	-0.92
3e	288^3	8	32	666,437	1.81	59.51h	13.68h	-0.89
3f	288^3	10	40	662,027	1.65	63.30h	15.59h	-0.88
3g	288^3	12	48	668,213	1.64	67.98h	15.75h	-0.86
3h	288^3	16	64	688,935	1.56	80.10h	18.70h	-0.81
3i	288^3	20	80	652,337	1.61	94.16h	28.11h	-0.77
3j	288^3	24	96	650,835	–	91.79h	23.80h	-0.79
5a	384^3	5	20	2,177,718	2.69	156.2h	29.42h	-1
5b	384^3	10	40	2,201,338	2.52	186.4h	38.54h	-0.75
5c	384^3	14	56	2,166,350	2.71	185.3h	37.00h	-0.83
5d	384^3	20	80	2,150,868	2.89	203.2h	44.23h	-0.81
5e	384^3	24	96	2,149,394	–	221.1h	44.52h	-0.78

Table 3.4: Scalability runs. All the runs used four processes per node. Column five shows the total number of mesh refinements performed by all processes during the whole run. The speedup column shows the relative increase in the overall speed of the adaptive run over the non-adaptive scalability runs in Section 2.7.5 which only used up to 20 nodes. The power law index S of the total simulation run wall clock time dependency $W \propto \text{Nodes}^S$ on the number of nodes is presented in a scalability column. Column (W1) shows the value of (Wall Clock Time)$\times$Nodes for each of the runs, where the wall clock time measures the time from the start of the run up to the finish. Column (W2) shows the same for the total wall clock timing spent within the PM-force calculation routine.

mains with the HC-mesh cells and for other reasons (see below).

The initial conditions are equivalent to those in the 800^3 run presented in Section 3.4, with the exception that we mapped fluctuations of density to the smaller number of particles. For example, we always used the same value for the softening length as in the 800^3 run as expressed in Mpc. In the result, the softening length is smaller as expressed in code units since the grid spacing is larger for the coarser particle and density mesh mapping the constant size (200 Mpc) simulation volume.

The parameter $\tilde{R}_{\rm c\,max}$ was set automatically by the procedure in Section 3.2.5. For 288^3 and 384^3 runs we have $\tilde{R}_{\rm c\,max} = 2.68$ and 2.59, leading to chaining mesh sizes of 107^3 and 148^3. We used Intel hyperthreading for all our runs (using two processes per processor, or four processes per computing node).

In Table 3.4, we present the main results for the scalability runs, analogous to the non-adaptive P^3M runs performed in Chapter 2, Table 2.4. Comparison of these two tables shows an apparent speedup achieved by computing forces adaptively. The magnitude of the speedup grows with the increase of the size of the problem $N_{\rm gr}$, which in our case corresponds to increasing comoving number density of the sampling of matter with particles.

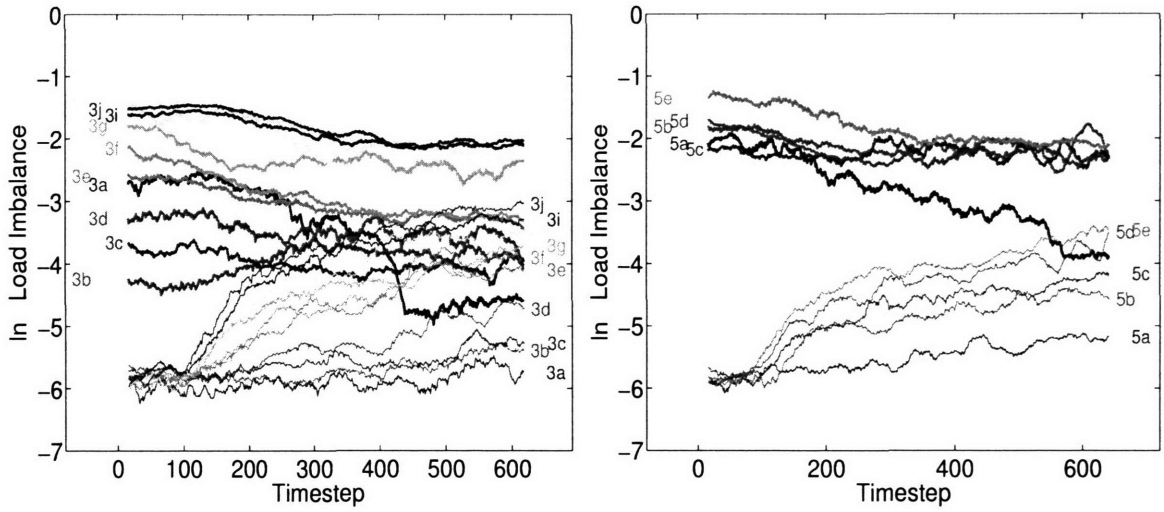

Figure 3-25: Instantaneous (heavy lines) and residual (thin lines) load imbalance as a function of timestep for Runs 3 ($N_{\rm gr} = 288^3$) and 5 ($N_{\rm gr} = 384^3$). The individual runs are labeled.

The overall simulation wall clock time speedup (shown in the Table) defined for each of the scalability runs as the ratio of the wall clock time needed to complete the same run by non-adaptive PP in Table 2.4 to the current measured time to evolve the problem using the adaptive PP. The speedup shown in the table is significant and is mainly due to the mesh refinements performed in our current runs. The total number of mesh refinements performed in each of the runs is shown in the table. We observe that the number is independent on the number of nodes used. This observation is in agreement with the fact that in our optimal mesh refinement scheme the number of processes $n_{\rm pr}$ is not used for making a decision on whether to do mesh refinement for a given cell.

Shown in column (W1) is the timing to evolve the problem all the way from the initial conditions to the final expansion factor $a = 1$, multiplied by the number of nodes used for the run. For a perfectly scalable code this number remains constant, independent of the number of nodes used, so that the wall clock time to complete the problem on a cluster of computers is inversely proportional to the number of nodes used $W \propto \text{Nodes}^{-1}$. In a large cosmological N-body problem we can not achieve perfect scalability, but we can achieve a good scalability, which can be quantified in terms of the *scalability slope* S defined for the runs presented in the Table 3.4 by

$$\frac{W}{W_0} \propto \left(\frac{\text{Nodes}}{\text{Nodes}_0}\right)^S , \tag{3.53}$$

where W is the wall clock time to evolve the whole simulation from the initial conditions to the final expansion factor $a = 1$, and the reference run values are marked with subscript 0.

In Figure 3-25 we present the measurements of the instantaneous and residual load

imbalance (as defined in Section 2.4.1) for each of the scalability runs. It is significant that due to the mesh refinements performed in adaptive PP, the workloads of each cell remain much less than the average workload per process and therefore the residual load imbalance does not grow significantly at high clustering remaining far below the average instantaneous load imbalance.

We can use equation (2.32) in order to determine an upper bound on the number of processes beyond which the problem is not effectively extendable. The maximum number of processes is given by dividing the total workload of a given scalability run at the final timestep by the maximum workload of a cell. We can use the discrete workload array (plots similar to Figure 2-17) to find this ratio. The procedure yields 320 of processes for the 288^3 runs and 360 for 388^3 runs. The same procedure for the set of the scalability runs with non-adaptive P^3M yields 57 and 62 processes respectively.

The scalability slope S measurements in Table 3.4 for run 3g is almost the same as the slope $S = -0.86$ measured in Chapter 2.

Using the adaptive-PP to speed up the PP-force computation, we have increased significantly the fraction of the wall clock time doing the PM-force calculation (see Section 3.3.1). The timing of the PM-force calculation, shown in Table 3.4, has worse scalability but is always much less than the PP-force computation timing, therefore its impact on the overall scalability is limited. In Section 3.3.1 introduced an improvement to our PM-force calculation algorithm, designed to achieve better scalability for computing the PM-forces. We can test the improvement by comparing the *PM-force scalability slope* measured using equation (3.53) with $W = W_{\rm PM}$, with the measurements of the slope in the runs performed before this improvement was introduced. We find that the PM-scalability slope has improved as $(-0.3) \rightarrow (-0.5)$ for the 288^3 runs and as $(-0.5) \rightarrow (-0.7)$ for the 384^3 runs.

Due to the optimization methods presented in Chapter 2 and the improvement presented in Section 3.3.1, the PM-forces are no longer dominant for big adaptive P^3M simulations. We do not expect that their timing will ever be influential in large adaptive-P^3M simulations. As we see from Table 3.4, the scalability of computing the PM-forces is improving, while its overall timing fraction decreases with increasing simulation size.

3.5 Conclusion

In this Chapter we developed an extension of the original parallel load-balanced scalable algorithm and implementation presented in Chapter 2 to include the adaptive computation of the short range or particle-particle forces, solving a significant problem arising at high clustering in any realistic cosmological N-body simulation.

The whole simulation volume is divided into a large number of so called coarse chaining mesh cells. Each process of the cluster of computers holds all the particles whose positions are within its local domain, defined as subset of the whole set of the chaining mesh cells. When clusters of particles form inside the simulation volume, the domains of some processes are occupied by bigger number of particles leading

to work load imbalance. The effective load balancing technique, leading to uniform distribution of the workload among the processes by readjustment of local domains performed each timestep, was introduced in Chapter 2. However, even at the highly load balanced regime, when all the processes are effectively using their CPU resources, very soon the particle distribution in an N-body problem become so clustered that it takes a divergent amount of time to do the PP part of force calculation.

In Section 3.2 we introduce an adaptive mesh refinement technique which takes place within the cells having very high number of particles and therefore making significant contribution to the total workload of the problem. Mesh refinement decreases the workloads of dense cells by orders of magnitude by transforming the cost of direct summation over the coarse chaining mesh cells to a composition of a Fast Fourier transform and direct summation over a finer mesh.

In Section 3.2.2 we introduce a method of avoiding the periodic boundary conditions, intrinsic to Fourier transform-based forces, for computing the PP-forces having isolated boundary conditions. The adaptive mesh refinement technique results in speeding up an N-body simulation by a large factor, that increases with the problem size.

There is only one main free parameter of mesh refinement — the size of the fine density mesh n_f. By analyzing the dependence of workload on the number of particles in Section 3.3.2, in Section 3.3.3 we arrive to a practical scheme to set this parameter individually for each cell resulting in nearly linear dependence of the adaptive-PP force calculation on the number of particles it contains, as opposed to the quadratic dependence when no adaptive-PP force calculation is used. The quasi-linear dependence of the cell workload greatly improves the load balancing of the problem.

We further reduce the workload by introducing additional techniques. In Section 3.3.1 we introduce non-blocking PM-communications that reduce the PM workload and improve its scalability to a large number of processes. In Section 3.3.4 we introduce the sorting procedure for particles applied each timestep and designed to reduce the FPP-workload by a large factor.

Doing mesh refinement does not result in a significant loss in the computed force accuracy. By setting the simulation parameters as shown in Section 3.2.4 we achieve any desired force accuracy for the gravitational forces computed using adaptive P^3M.

In Section 3.4 we perform simulations identical to the ones described in Chapter 2 and compare the improvement in the performance. In the 800^3 particle ΛCDM simulation a speedup of more than a factor of 5 was achieved for the total wall clock time needed to advance the gravitational N-body problem to expansion factor $a = 1$.

In Section 3.4.1 it is shown that simulations performed with our code are scalable with the scalability slope $S \approx 0.8$. Also it is shown that the upper limit on the maximum number of processes, beyond which the problem is unscalable, has increased by an order of magnitude. It is shown that better scalability of the PM-force component is achieved by introducing non-blocking PM-communications in Section 3.3.1.

We seek further improvement by implementing adaptive timesteps leaving it for future work.

Chapter 4

Simulations of Small-Scale Structure

4.1 Introduction

The particle content of dark matter is so far unknown because there has been no solid detection of dark matter by means other than its gravitational interaction. In the past, many new particles (such as the positron) were first predicted theoretically before they were detected in experiments. Current theoretical models of particle physics predict the existence *weakly interacting dark matter particles* (*WIMPs*, or *neutralinos*), while others predict *axions* (see the review in [33]). These two are the leading candidates for the dark matter particles.

The dark matter in the universe can be considered as a cold collisionless fluid, whose dynamics is described as the evolution of a three dimensional surface in six-dimensional phase space (as will be shown in Section 4.2.3). Cold dark matter *caustics* are surfaces of infinite density formed by the projection of the distribution function of fluid elements of the three dimensional surface in the phase space onto the three spatial dimensions. Caustics form in the very early stage of the evolution of cold dark matter, as early as redshift ~ 100. Later on, they develop into highly complicated stable topological structures.

The formation of caustics in cold dark matter was first predicted analytically by [25] and [5], where symmetric models of the dark matter distribution are considered, and the equations are given for the matter distribution in both physical and phase space.

In the spherically symmetric analytical model considered in [25] and [5], the caustics form starting from a spherically symmetric and uniform overdensity. This overdensity evolves, forming a set of expanding spherical shells of infinite density — caustics — in physical space.

These singularities, called *fold* singularities, are the simplest case of topological singularities. Even within the framework of the analytical models, more complicated caustics structures are possible, such as caustic rings [47]. The whole classification of projective phase space singularities is given in [50].

In the more complicated world of the real universe, one expects that the smallest size caustics are formed as sets of spherical shells around different points of initial small-scale overdensities throughout the universe. Larger scale density fluctuations form a pattern whose evolution is perhaps similar, except that it is far more complicated since it contains contains substructure of small-scale mini- regions of caustic formation. These large scale caustics form a higher level in the spatial hierarchy.

At this time, there are about 20 ongoing experiments trying to detect neutralinos by the recoil momentum they give to nuclei with which they collide. There is no accepted detection of dark matter at the time of this writing, only the upper limits on the interaction cross section.

If the dark matter is composed of neutralinos, caustics and other small-scale structure may produce a significant contribution to the dark matter experiments and therefore change the conclusions on experimental limits for neutralino cross sections. In direct detection experiments the signal is proportional to the local density of dark matter at the point of the detector position. The event rate is highly variable depending on whether or not a caustic crosses the detector.

In addition to the direct detection experiments, there are efforts to make an indirect detection of dark matter due to WIMP annihilation, which would produce detectable products in the form of gamma rays, electrons and positrons. The contribution of caustics to the measured signal is especially high for WIMP annihilation experiments because the rate is proportional to the square of the local density.

So far, experiments have used very simple models of the WIMP distribution to set limits on the neutralino-nucleon cross section. In [1] an isothermal sphere dark matter distribution filling the Galaxy was assumed as the true dark matter distribution. In [4] it was assumed that dark matter is characterized by a quasi-Maxwellian distribution in the Galactic plane with the escape velocity ~ 650 km s^{-1}. The matter density in both models is locally smooth.

Despite strong arguments in favor of caustic substructure, following the original work by [25] and [5] the presence of caustics has been largely ignored, due to several reasons. First, in the real universe, symmetry of structure is not a requirement and it is impossible to work out an accurate analytical model of nonlinear structure formation without assuming highly symmetric perturbations. Secondly, caustics are not seen in the typical cosmological simulations of dark matter.

Perhaps the fact that caustics are not observed in typical N-body simulations led some to believe [30] that caustics are not formed or are easily destroyed by evolution in the real universe. It is very important to remember that the discreteness of sampling of the real physical particles with massive simulation particles in an N-body simulation leads to effects that are physical in nature but are not necessarily characteristic of the real dark matter particles. In the real universe the matter is sampled with real dark matter particles, not the simulation particles, and even the heaviest plausible dark matter particle candidate, a 500 GeV neutralino [19], is 6.2×10^{62} times less massive than the simulation particles in a high resolution 1024^3 N-body simulation with 200 Mpc box size.

Structure forming on scales smaller than a simulation particle is not resolved by a simulation and as a consequence it is usually forgotten. This mass range is however

perfectly covered with real particles where the formation of substructure is allowed. As we will discuss below, the only physical effect that can lead to the destruction of caustics is heating due, for example, to two-body relaxation. Such collisional relaxation leads to a finite temperature that leads to smearing of caustics [48]. Of course, in real cold dark matter caustics, the density in physical space does not become infinite. Due to sampling with a huge but still finite number of neutralinos and their small but finite temperature, the peak density reaches a finite but very high value. As we will see below, collisional effects and the temperature are so small for real neutralinos that the effects leading to the destruction of caustics are negligible. In the real universe, the three dimensional structure of the matter distribution in phase space is preserved. We will focus our following discussion on neutralinos, for which a model of the small scale initial perturbations has been worked out.

In this work we perform a set of simulations with the aim of describing the formation and evolution of dark matter caustics. The caustics are easily resolved as a result of adjusting the resolution of the simulation to the level required in order to see their formation. The simulations demonstrate the formation of caustics in an N-body simulation when the smallest caustics are fairly sampled by the simulation particles. The simulations follow the formation of caustics deep into the stage of nonlinear evolution. Halo mergers result in a highly complicated caustic topology. The only critical difference of our simulation from a typical cosmological N-body simulation is the dark matter mass resolution. In a 1024^3 particle N-body simulation with 200 Mpc box size and $\Omega_m = 0.3$, the mass of a simulation particle is 3.1×10^8 $M_\odot$, while in the simulation described in Section 4.3.3, one particle weighs only 3.1×10^{-12} $M_\odot$. The mass resolution in our simulation is so high that is sufficient to resolve the effects of free streaming and collisional damping in real dark matter. The sizes of the simulation boxes used here are much smaller than in the typical N-body simulations.

4.2 Analytical Theory of Dark Matter Caustics

Caustics are rigorously defined for a perfect cold collisionless fluid (CCF). The properties of caustics in the d-dimensional physical space are best understood by looking into the evolution of their structure in phase space. Below we offer both Lagrangian and Eulerian descriptions for evolution of a CCF. We consider the structure in both the continuous limit and the discrete particle approach of an N-body simulation.

4.2.1 Dark Matter as a Cold Collisionless Fluid

The theoretical models for neutralinos have many free parameters and their particle mass is poorly constrained. However, from astrophysical and experimental arguments, the neutralino mass is believed to be within the range of $m_\chi = 6 - 500\,\mathrm{GeV}$ [11], [19]. The cross section of weak interaction for neutralinos is extremely small, which makes their direct detection hard.

In the early universe, when the number density of neutralinos in the universe and their temperature were sufficiently high, neutralinos efficiently annihilated into other

particles. When the annihilation rate dropped below the Hubble expansion rate, the neutralinos effectively stopped annihilating and their comoving abundance became constant (*chemical decoupling* or *freeze-out*) at $T = T_{\rm cd} \sim m_\chi/25$. Although the average present rate of interaction of neutralinos is so small that any given neutralino is unlikely to interact with any other particle in a Hubble time, some of the leftover neutralinos should interact occasionally, producing directly detectable annihilation products. The annihilation cross section of neutralinos is about the same as the collisional cross section with themselves and the other weakly interacting particles. At the time of chemical decoupling, due to the heavy mass of neutralinos, their number density was much smaller than the number density of other weakly interacting particles, such as neutrinos, electrons and muons. Therefore after the chemical decoupling of neutralinos has occurred, neutralinos were still kept in thermal equilibrium by elastic collisions with these particles. At some point, *kinetic decoupling* occurred when the elastic scattering rate dropped below the Hubble expansion rate. It is estimated in [46] that, depending on the particle model, the temperature of the kinetic decoupling is $10 - 40\,\mathrm{MeV}$, at which point the neutralinos were definitely non-relativistic. After this epoch, one may consider the dark matter dynamics as *collisionless* to a very good approximation.

Since the neutralinos were in thermal equilibrium with the thermal plasma before kinetic decoupling, and afterwards the neutralino temperature redshifted as $(1+z)^2$, we estimate the present neutralino temperature as $T_{\rm kd}(1+z_{\rm kd})^{-2} \approx 2.1 \times 10^{-11} \times (30\,\mathrm{GeV}/m_\chi)\,\mathrm{K}$, yielding a thermal velocity of $0.006 \times (30\,\mathrm{GeV}/m_\chi)\,\mathrm{cm\,s^{-1}}$. On the other hand, the mean circular velocity in the vicinity of the Sun 220 km s^{-1} is orders of magnitude greater than the thermal velocity of neutralinos. It is a very good approximation to neglect the neutralino temperature.

The mean interparticle distance for neutralinos in the universe is estimated by dividing the average dark matter density of the universe by the mass of the neutralino and taking the inverse cube root. In [29], it is estimated that the smallest clumps of dark matter particles enter the nonlinear regime at a redshift 60, at which point, assuming the dark matter particles consist of WIMPs of mass 100 GeV, the proper interparticle distance was 6.7 cm. At that redshift, they estimate the matter power spectrum cutoff at small scales was at the mass scale $10^{-6}\,\mathrm{M}_\odot$, corresponding to a proper smoothing scale of 9.3×10^{16} cm. The interparticle distance is therefore orders of magnitude smaller than the length scales at which the CDM velocity field changes. The ratio of smoothing scale to interparticle distance is enormous, thus the dynamics of the dark matter can be accurately described as the dynamics of a continuous fluid.

Summarizing the above arguments, the cold dark matter can be described very well as a *cold collisionless fluid* (CCF).

4.2.2 Lagrangian and Eulerian Description of Perfect Fluid

In the following we will use notation $(\mathbf{q}, \mathbf{u})$, and $(\mathbf{x}, \mathbf{v})$ for the coordinate-velocity pair in Lagrangian and Eulerian spaces, where $\mathbf{q}$ and $\mathbf{x}$ are comoving distances, t is conformal time, $\mathbf{u} = d\mathbf{q}/dt$ and $\mathbf{v} = d\mathbf{x}/dt$.

Each fluid element of the CCF is given a unique fixed set of Lagrangian coor-

dinates $\mathbf{q} = (q^0, q^1, q^2)$. The kinematics of the CCF is described as a continuous mapping, giving the coordinates in Eulerian space of each of the fluid elements as a function of time t and their Lagrangian $\mathbf{q}$ coordinates

$$\mathbf{x}(t) = \mathbf{X}(t, \mathbf{q}) \tag{4.1}$$

This mapping defines the evolution of a curvilinear mesh of Lagrangian coordinates in Eulerian space.

Note that this description is invalid for matter that is not described as CCF. Indeed, consider a high temperature gas for example, each small subvolume of a gas consists of particles moving in different directions. Two gas particles initially infinitesimally close to each other may finish up at a finite time at completely different Eulerian coordinates, for which case the description (4.1) is invalid.

As was shown above, neutralino dark matter can be described as a cold collisionless fluid, so that the analytical techniques resulting from equation (4.1) are in this case realized in practice. In the linear theory of dark matter perturbations, the density fluctuation amplitude δ grows with the increase of the expansion factor of the universe as $\delta \propto a$ when the universe is matter-dominated. At much earlier times the matter is essentially uniform. It is then convenient to label particles by their initial positions on a uniform grid in comoving coordinates

$$\lim_{t \to 0} \mathbf{X}(t, \mathbf{q}) = \mathbf{q} \tag{4.2}$$

Since particles do not move in Lagrangian space, the matter density ρ_0 in Lagrangian space is constant and is equal to the initial comoving matter density in Eulerian space.

With the increase of the cosmic time, Eulerian positions deviate from their initial values. As a result, density perturbations arise in Eulerian space. If the positions are sufficiently evolved, the linear theory of matter perturbations no longer applies, and the evolution of density perturbations deviates from $\delta \propto a$. At this stage there appear perturbations whose relative amplitude has grown above unity. Indeed, such a perturbation may grow to infinite density at certain points within the volume. To see how this happens, let us just write the expression for the density of the fluid in terms of the Jacobian of the Eulerian mapping (4.1)

$$\rho(t, \mathbf{x}) = \sum_{\mathbf{q}_j} \rho(\mathbf{X}(\mathbf{t}, \mathbf{q_j})) = \rho_0 \sum_{\mathbf{q}_j} J^{-1}(t, \mathbf{q}) \text{ , where } \quad J(t, \mathbf{q}) \equiv \det \left| \frac{\partial \mathbf{X}(t, \mathbf{q}_j)}{\partial \mathbf{q}} \right| , \tag{4.3}$$

where $\mathbf{q}_j$ are all values of Lagrangian coordinates such that $\mathbf{x} = \mathbf{X}(t, \mathbf{q}_j)$. In general one point $\mathbf{x}$ in Eulerian space can have one or more roots in Lagrangian space satisfying the last equation. A point in Eulerian space is therefore characterized by its *multiplicity*, or the number of Lagrangian space prototypes it has. We impose the restriction that the mapping $\mathbf{X}(t, \mathbf{q})$ be continuous in both t and $\mathbf{q}$, which is a valid assumption for motion under gravity.

A *caustic* occurs in Eulerian space where the multiplicity changes. Such changes

occur across a surface in space where the Jacobian vanishes

$$\text{caustic}: \quad J(t, \mathbf{q}) = 0 \,. \tag{4.4}$$

Caustics form a lower dimension subset of Eulerian space. Due to the large freedom for transformation (4.1) allowed by Hamiltonian evolution, the caustics in general may form a highly complicated topological structure in Eulerian space, especially for matter distributions lacking symmetry.

Depending on the distribution of the CCF in phase space, various types of singularities are possible in physical space. The full classification of the singularities is given in [50]. The simplest of them and also the most frequently forming under typical evolution is called a *fold caustic*. In a d-dimensional physical space the matter in a fold caustic forms a $(d-1)$-dimensional surface.

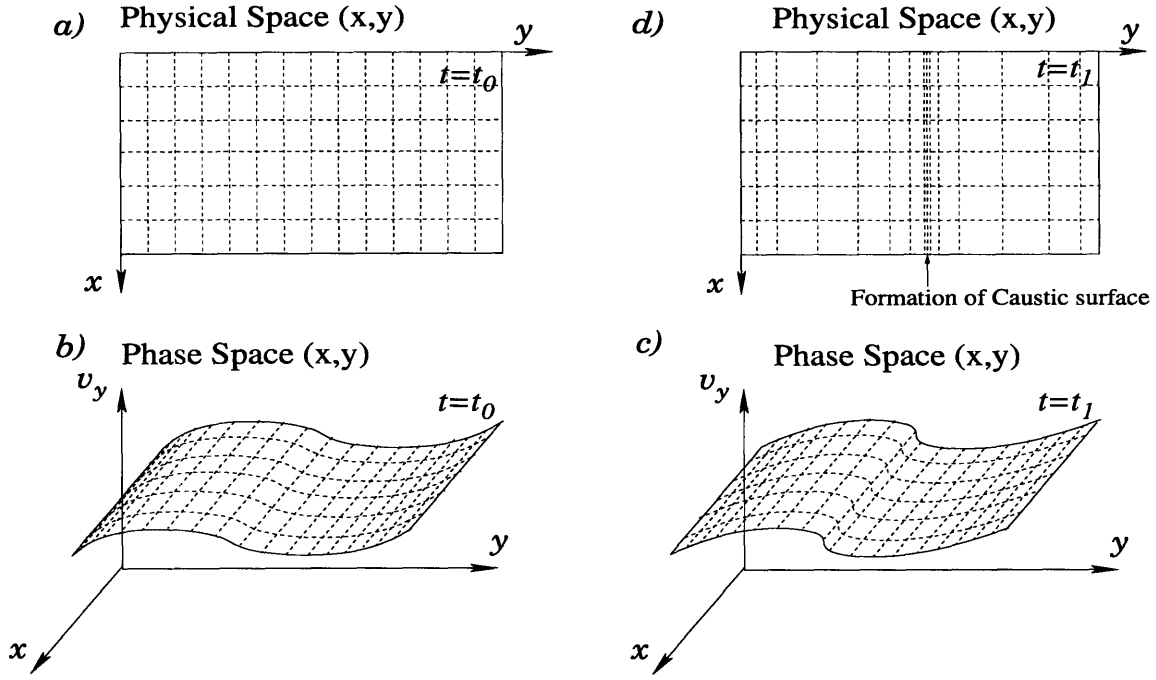

Figure 4-1: The origin of a caustic fold singularity in an example initial matter distribution in phase space. The dashed lines are lines of constant Lagrangian coordinates. Panels a) and b) are simple projections of the phase space density [panels c) and d)] along the velocity dimension v_y. Caustics form at time $t = t_1$ as a infinite density singularity in the projection from phase space onto physical space [panels c) and d)].

One can illustrate the formation of a fold-type caustic, by considering an initially spatially uniform two dimensional matter distribution in Eulerian coordinate space, as shown in Figure 4-1. Let us consider a plausible situation when at some point within the flow the velocity gradient along y direction is negative and largely exceeds the velocity gradient along the x direction at the initial time t_0, in which case we can locally neglect the v_x velocity component. If the tidal acceleration within the

flow is small, following the evolution to the later time $t = t_1 > t_0$, the positions of particles are advanced along the y-direction, increasing the waviness in the phase-space distribution. The development of folds in phase space leads to the formation of density singularities (caustics) in the physical space.

4.2.3 Phase Space Interpretation of Caustics

It is important to consider the cold collisionless fluid in phase space because the caustics that show up as density singularities in the physical space are no longer singularities in the phase space. In fact, the physical space singularities are simply the projection effect from the $2d$-dimensional phase space to the d-dimensional physical space.

The physical space in both Lagrangian and Eulerian description of the fluid is defined as a subset of the d-dimensional real space $\mathbb{R}^d$. Given the mapping (4.1), the extension to Eulerian phase space (a subset of $\mathbb{R}^{2d} = \mathbb{R}^d \times \mathbb{R}^d$) is performed by taking an additional d velocity-momentum components by differentiation of the mapping $\mathbf{X}(t, \mathbf{q})$ with respect to the time variable

$$\mathbf{v} \equiv \frac{d\mathbf{x}(t, \mathbf{q})}{dt} = \frac{\partial \mathbf{X}(t, \mathbf{q})}{\partial t} \equiv \mathbf{V}(t, \mathbf{q}) \,. \tag{4.5}$$

By the definition of Lagrangian space, $\mathbf{q}$ does not change for a fluid element.

Let us consider a differentiable manifold A within the d-dimensional real space $\mathbb{R}^d$. The manifold A is said to have the natural valued *dimensionality* m within the space $\mathbb{R}^d$, if and only if for any vector $\mathbf{a}_0$ within A there exists a set of local basis vectors $\mathbf{e}_i$ such that for any real $\epsilon > 0$ there exists $\eta > 0$ such that for any $\mathbf{a} \in A$, in the η-vicinity of $\mathbf{a}_0$, $|\mathbf{a} - \mathbf{a}_0| < \eta$, there exists a set of real numbered components α^i, $i = 1 \ldots m$, satisfying

$$|\,\mathbf{a} - \mathbf{a}_0 - \sum_{i=1}^{m} \alpha^i \mathbf{e}_i| < \epsilon \,. \tag{4.6}$$

In order to apply our definition of dimensionality to phase space, let us introduce a metric to measure distances in phase space. Consider a general $2d$-dimensional real vector space $\mathbb{R}^{2d}$. The interval ds is defined for vectors in this space as

$$ds^2 = \eta_{ij} da^i da^j \tag{4.7}$$

where a^i are the components of vector $\mathbf{a} \in \mathbb{R}^{2d}$. For both Lagrangian and Eulerian descriptions, a metric can be defined in the $2d$-phase space $\mathbb{R}^{2d}$ as

$$\eta_{ij} = \begin{cases} \delta_{ij} & i = 1 \ldots d \\ T\delta_{ij} & i = d+1 \ldots 2d \end{cases} \tag{4.8}$$

where T is a constant parameter having the physical dimension of time necessary in order to bring the coordinates and velocities to the same units.

According to this definition, the dimensionality of the volume filled by the CCF

in Lagrangian $2d$-dimensional phase space of Lagrangian position $\mathbf{q}$ and velocity

$$\mathbf{u} \equiv \frac{d\mathbf{q}}{dt} = 0$$

is d. Indeed, for any fixed $\mathbf{q_0} \in \mathbb{R}^{2d}$ within CCF, its spatial components are just a set of real numbers $q_0^1, \ldots, q_0^d \in \mathbb{R}$, and the velocity components $u_0^1 = q_0^{d+1}, \ldots, u_0^d = q_0^{2d}$ are all zero. Let us define a set of d local basis vectors for this point precisely in the directions of the spatial axes. For any real positive numbers ϵ and η and any vector $\mathbf{q}$ within the CCF (zero velocity components) in the η vicinity of another arbitrary fixed vector $\mathbf{q_0}$ within the CCF apparently satisfies the condition (4.6) by setting $\alpha^i = q^i - q_0^i$ [we are using $\mathbf{q}$, $\mathbf{q_0}$ in place of $\mathbf{a}$, $\mathbf{a_0}$ in (4.6)].

We can prove that the CCF in Eulerian description at a given time t_0 forms a d-dimensional subset of the whole Eulerian phase space $\mathbb{R}^{2d}$. Consider any fixed point $\mathbf{x_0}$ of the CCF at time t_0 and let $\mathbf{q_0}$ be its Lagrangian position so that $\mathbf{x_0} \equiv \mathbf{x}(\mathbf{q_0}) = \mathbf{X}(t_0, \mathbf{q_0})$. The fluid velocity $\mathbf{v_0}$ at this Lagrangian point is given by equation (4.5). Let us define the $2d$-dimensional vector $\mathbf{a_0}$ as the point in phase space whose position is given by the components of vectors $\mathbf{x_0}$ and $\mathbf{v_0}$. We adopt a notation $\mathbf{a_0} \equiv (\mathbf{x_0}, \mathbf{v_0})$ in general, for listing the components of a $2d$-dimensional quantity $\mathbf{a_0}$ whose first d components are given by the d components of the first vector $\mathbf{x_0}$ in the brackets, while the last d components being equal to the d components of the second vector $\mathbf{v_0}$. Since the mapping (4.1) is differentiable, the Eulerian coordinate and velocity components of the fluid in the vicinity of point $(\mathbf{x_0}, \mathbf{v_0})$ of the phase space may be expanded in Taylor series as

$$\begin{aligned} \mathbf{x}(\mathbf{q}) &= \mathbf{x}(\mathbf{q_0}) + \frac{\partial \mathbf{X}}{\partial \mathbf{q}} \cdot (\mathbf{q} - \mathbf{q_0}) + O(\mathbf{q} - \mathbf{q_0})^2 \\ \mathbf{v}(\mathbf{q}) &= \mathbf{v}(\mathbf{q_0}) + \frac{\partial \mathbf{V}}{\partial \mathbf{q}} \cdot (\mathbf{q} - \mathbf{q_0}) + O(\mathbf{q} - \mathbf{q_0})^2 \, . \end{aligned} \tag{4.9}$$

Defining the Eulerian phase space position of Lagrangian point $\mathbf{q}$ as $\mathbf{a} \equiv (\mathbf{x}(\mathbf{q}), \mathbf{v}(\mathbf{q}))$, choosing the local basis of d vectors at $\mathbf{a_0}$ as $\mathbf{e}_k \equiv (\partial \mathbf{X}/\partial q^k, \partial \mathbf{V}/\partial q^k)$, defining the $2d$ components as $\alpha^k \equiv (q^k - q_0^k, q^k - q_0^k)$, where $k = 1 \ldots d$, and using the definition (4.7) and (4.8) of metric, equation (4.6) is written as

$$\begin{aligned} |\, \mathbf{a} - \mathbf{a_0} - \sum_{i=1}^{m} \alpha^i \mathbf{e}_i | = \Bigg[\Big| \mathbf{x}(\mathbf{q}) - \mathbf{x}(\mathbf{q_0}) - \tfrac{\partial \mathbf{X}}{\partial \mathbf{q}} \cdot (\mathbf{q} - \mathbf{q_0}) \Big|^2 + \\ T^2 \Big| \mathbf{v}(\mathbf{q}) - \mathbf{v}(\mathbf{q_0}) - \tfrac{\partial \mathbf{V}}{\partial \mathbf{q}} \cdot (\mathbf{q} - \mathbf{q_0}) \Big|^2 \Bigg]^{1/2} = O(\mathbf{q} - \mathbf{q_0})^2 \, . \end{aligned} \tag{4.10}$$

We have therefore proven that the CCF in phase space satisfies the definition (4.6) of a d-dimensional subspace of the whole $2d$ dimensional phase space $\mathbb{R}^{2d}$. In other words, the phase space positions of fluid elements of the CCF are locally expanded in the basis of d vectors in $2d$-dimensional space, therefore we have proven that the CCF is a d-dimensional set in the phase space.

The dimensionality of the phase space distribution sets the properties of the phase

space distribution function. We define $f_{\rm L}(t, \mathbf{q}, \mathbf{u})$ and $f_{\rm E}(t, \mathbf{x}, \mathbf{v})$ to be the phase space distributions in Lagrangian and Eulerian spaces. These functions, when integrated over the whole phase space give the total mass of particles. The Lagrangian phase space distribution in the space of Lagrangian coordinates and velocity $\mathbf{u}$ is simply defined as

$$f_{\rm L}(t, \mathbf{q}, \mathbf{u}) = \rho_0 \, \delta^3(\mathbf{u}) \, , \tag{4.11}$$

where ρ_0 is the initial background density, usually expressed in terms of the number density as $\rho_0 = m n_0$.

The phase space distribution function in Eulerian space differs from $f_{\rm L}$ by a factor of the Jacobian of the transformation (4.3),

$$f_{\rm E}(t, \mathbf{x}, \mathbf{v}) = \rho_0 J^{-1} \, \delta^3(\mathbf{v} - \dot{\mathbf{X}}(t, \mathbf{q})) \, , \tag{4.12}$$

where $\mathbf{q}$ is the unique root of equation $\mathbf{x} = \mathbf{X}(t, \mathbf{q}_j)$.

The density of matter in physical space is obtained from Eulerian phase space density by the projection of phase space density along the velocity using equation (4.3),

$$\rho(t, \mathbf{x}) = \int f_{\rm E}(t, \mathbf{x}, \mathbf{v}) d^3\mathbf{v} = \rho_0 \sum_{\mathbf{q}_j} J^{-1}(t, \mathbf{q}_j) \tag{4.13}$$

where $\mathbf{q}_j$ are all the unique roots equation of $\mathbf{x} = \mathbf{X}(t, \mathbf{q})$. The above equation coincides with (4.3).

4.2.4 Caustics in Zel'dovich approximation

Zel'dovich [53] gave the solution for the mapping (4.1), exact in the linear regime and still valid early when the perturbations are not small

$$\mathbf{r} \equiv a\mathbf{x}(\mathbf{q}, t) = a(t)[\mathbf{q} + D_+(t)\psi(\mathbf{q})] \, , \tag{4.14}$$

where $\psi(\mathbf{q})$ is the initial displacement field and $D_+(t)$ is the growing mode of the linear density contrast. In a universe consisting of nonrelativistic matter, vacuum energy and no other types of matter, for perturbations whose comoving wavenumber obeys $c^{-1}H \ll k \ll k_J$, where k_J is the comoving Jeans wavelength, the growing mode is given by

$$D_+(a) = H(a) \int_0^a \frac{da}{(Ha)^3} \, , \tag{4.15}$$

as expressed in terms of the expansion factor rather than cosmic time. The transformation (4.14) is an identity for $t = 0$, in agreement with (4.2).

The comoving displacement $\Delta\mathbf{x}$ of a particle from its Lagrangian coordinate at $t = 0$ is related to the proper peculiar velocity $\mathbf{v}$ by

$$\mathbf{v} = a\frac{d\mathbf{x}}{dt} = a\frac{dD_+(t)\psi(\mathbf{q})}{dt} = \frac{1}{D_+}\frac{dD_+}{d\tau}\Delta\mathbf{x} \tag{4.16}$$

where equation (4.14) was used twice. The last equation is used in the linear regime of matter perturbations in order to find the velocities of particles when their positions are known.

Writing a tensor of deformation of Eulerian fluid element under the transformation (4.14)

$$T^{ik} = \frac{\partial x^i}{\partial q^k} = \delta^{ik} + D_+(t)\frac{d\psi^i}{dq^k} \tag{4.17}$$

we can write the instantaneous differential displacement as

$$dx^i = T^{ik}(\mathbf{q}, t)\, dq^k \tag{4.18}$$

Now, since the initial velocity field is given as a gradient of a potential, it follows that $d\psi^i/dq^k = d\psi^k/dq^i$, and therefore $T^{ik} = T^{ki}$ (suppressing the arguments of T^{ik} for brevity). A symmetric matrix $d\psi^i/dq^k$ has a set of real eigenvalues $\lambda^i_{\mathbf{q}}$ ($i = 1, 2, 3$), and corresponding set of orthogonal eigenvectors. The whole deformation tensor has eigenvalues $1 + D_+(t)\lambda^i_{\mathbf{q}}$ and the same eigenvectors. Decomposing both the displacement vectors $d\mathbf{x}$ and $d\mathbf{q}$ in terms of the components along the eigenvectors, we find that the physical volume of the fluid element changes as

$$d^3\mathbf{x} = [(1 + D_+(t)\lambda^1_{\mathbf{q}})(1 + D_+(t)\lambda^2_{\mathbf{q}})(1 + D_+(t)\lambda^3_{\mathbf{q}})]\, d^3\mathbf{q} \tag{4.19}$$

The value in the square brackets is the Jacobian determinant $J(t, \mathbf{q})$ of the Eulerian transformation.

The condition for the caustic formation (4.4) requires that at least one of the eigenvalues $\lambda^i_{\mathbf{q}}$ be negative, since the function $D_+(t)$ is positive and monotonically growing. The initial velocity flow $\boldsymbol{\psi}(\mathbf{q})$ is a random field, implying that that the eigenvalues $\lambda^i_{\mathbf{q}}$ are random variables. Their probability distribution for a particular power spectrum of density distribution having a cutoff at small physical scales is given by [17]. The distribution, when expressed in our notation, has a peak in the negative domain of the eigenvalues. It is extremely likely that for a most general plausible power spectrum, at least one of the eigenvalues will be negative for most of the fluid elements, leading to a caustic as soon as the corresponding term in the square brackets in (4.19) becomes zero.

The Zel'dovich approximation shows that caustics form under a general type of the initial velocity distribution, if it continuous and differentiable.

4.2.5 Predictions from Self-Similar Analytic Models

Precise analytical solutions for halo evolution were developed in [25] and [5] for matter distributions that have special symmetries in the initial conditions. The asymptotic solutions are found for time $t \gg t_i$, where t_i is the time at which the initial conditions are specified.

A particle initially having zero peculiar velocity stays at a constant comoving position until it begins to fall towards the halo. In proper coordinates the particle at fixed comoving coordinates moves radially away due to Hubble expansion until

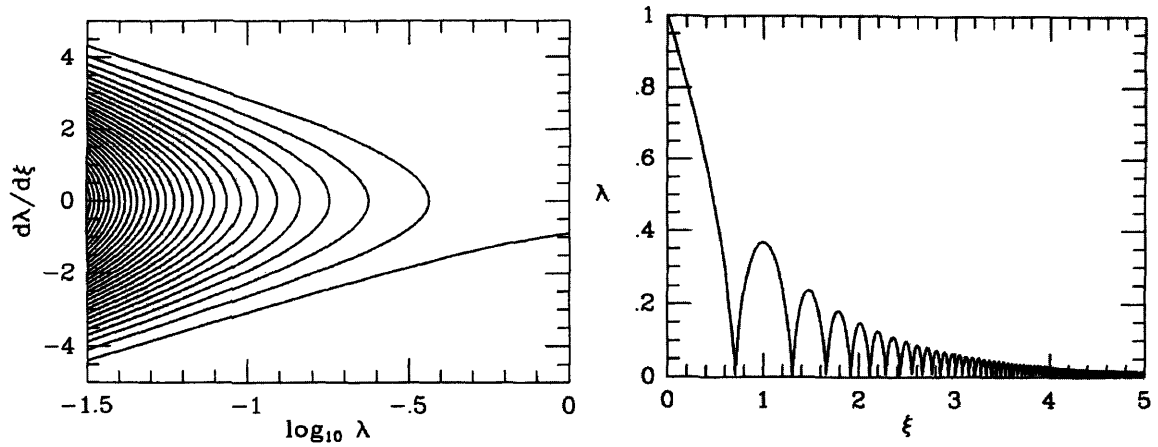

Figure 4-2: Left: The dependence of the dimensionless velocity $d\lambda/d\xi$ on the dimensionless coordinate ξ of all particles in the system at a given time, i.e. the phase space portrait of the self-similar model. In addition, the same pattern is followed by a each individual particle's trajectory in these coordinates. Right: The trajectory of a particle in physical space. Reproduced from [5] with the permission of E. Bertschinger.

the peculiar velocity component towards the halo becomes sufficiently high so that at some moment the particle turns around and starts falling towards the halo. This time is called the particle *turnaround time*, and the proper distance moved at that time is called the *turnaround radius*.

If there are no length scales present in the perturbations, its evolution is self-similar. As shown in Figure 4-2, the phase space portrait of the system remains the same at any time t in the coordinate system with the axes rescaled as $\xi \equiv \ln(t/t_{\rm ta})$ and $\lambda \equiv r/r_{\rm ta}(t)$, where $t_{\rm ta}$ is the particle's turnaround time, and $r_{\rm ta}(t)$ is the current turnaround radius. Due to the self-similarity, the Left plot in Figure 4-2 has a double meaning: it shows not only the positions of all the particles in the volume at a given time, but also the phase space trajectory of any given particle.

The initial conditions in [5] are given by a spherically symmetric tophat perturbation. In [25] the initial perturbation is a more general power-law density perturbation profile parametrized by an exponent ϵ_n giving the density perturbation as a function of separation from the center of perturbation as follows

$$\frac{\delta\rho}{\rho} \propto \frac{\delta M}{M} \propto r^{-3\epsilon_n} . \tag{4.20}$$

The top-hat perturbation considered in [5] corresponds to the case $\epsilon_n = 1$. Let us notice that the small scale structure of the initial conditions is unimportant for the late evolution of the halo and is relevant only for the initial infall. The details of the local mass distribution can only provide force gradients which average to zero at large separations. After the initial perturbations collapse, the halo asymptotically approaches the self-similar solution. However, if there are significant long-wavelength

perturbations, the evolution will not be self-similar.

In all the analytical halo evolution models, only a single perturbation is considered within the uniform background. In reality, the initial conditions for matter distribution in the universe are described as a Gaussian random field and are characterized by the power spectrum of density fluctuations $P(k) \propto k^n$, where k is the wavenumber of the matter perturbations. The question of how a typical halo produced as the result of the self-similar collisionless evolution looks like, given an initial power spectrum, was considered by [32], who provide the relation

$$\epsilon_n = \frac{3+n}{3} \tag{4.21}$$

between the initial density perturbation slope ϵ_n of equation (4.20) and the power spectrum slope. In particular, the white noise slope $n = 0$ yields $\epsilon_n = 1$, and the small-scale CDM power spectrum with $n = -3$ yields $\epsilon_n = 0$. Equation (4.21) is an approximation based on the average profiles of peaks in a Gaussian random field. A peak of the fluctuations in the Gaussian random field in general however has a very complicated profile not describable by a power law. For this reason, the value of ϵ_n given by (4.21) is imprecise, and in reality changes not only from peak to peak within the simulation box, but also as a function of the radial distance from a single peak.

The solutions in [25] provide the particle orbits and other important halo evolution parameters as a function of ϵ_n. In particular, the proper turnaround radius $r_{\rm ta}(t)$ is given as a function of time as

$$r_{\rm ta}(t) \propto t^{2/3+2/(3n_{\rm sym}\epsilon_n)} \tag{4.22}$$

where $n_{\rm sym} = (1, 2, 3)$ for the cases of planar, cylindrical and spherical symmetries. Equation (4.22) provides a sensitive dependence of the turnaround radius evolution as a function of ϵ_n. In general, the slope of the dependence of the turnaround radius on expansion factor becomes more shallow as the symmetry increases.

In a numerical simulation a number of factors complicates halo evolution, such as the absence of spherical symmetry, periodic boundary conditions, finite sampling of the power spectrum, limited dynamic range of the force and mass resolution. The sensitivity of the dependence of the turnaround radius on ϵ_n makes it difficult to analyze the source of any deviation from the self-similar model, which could be due to any of the above mentioned effects, in addition to the inaccuracy of the relationship (4.21) for ϵ_n.

It is more interesting to measure statistics expected to be less sensitive to ϵ_n. For example, the spherically averaged density profile should obey

$$\rho(r) \propto r^{-\gamma}\,, \tag{4.23}$$

where

$$\begin{aligned} \gamma &= 2, \quad && \text{for} \quad \epsilon_n \leq 2/3 \\ \gamma &= \tfrac{9\epsilon_n}{1+3\epsilon_n} && \text{for} \quad \epsilon_n \geq 2/3 \end{aligned} \tag{4.24}$$

For the cases of $\epsilon_n = (0, 1)$ these relations yield $\gamma = (2, 9/4)$. A deviation in the

measured density slope can not be due to the inaccuracy of the measurements in ϵ_n values, since the slope does not depend on this parameter sensitively.

On the other hand, if the parameter ϵ_n changes significantly, and crosses the value $\epsilon_n = 2/3$ at some radial separation within the perturbation, the power slope may show a break as it goes one value of γ to the other.

Another quantity showing discrete dependence on ϵ_n [25] is the evolution of the apoapsis (maximum separation) distance r_a of a fixed particle orbit with time, written in terms of the parameter q as

$$r_a \propto t^q \,, \tag{4.25}$$

where

$$\begin{array}{lll} q = \frac{3\epsilon_n - 2}{9\epsilon_n} & \text{for} & \epsilon_n \leq 2/3 \\ q = 0 & \text{for} & \epsilon_n \geq 2/3 \,. \end{array} \tag{4.26}$$

For the cases of $\epsilon_n = (0, 1)$ these relations yield $q = (-\infty, 0)$, meaning respectively that the maximum radius of a particle orbit goes quickly to zero or a constant.

It is interesting to consider the cylindrical symmetry case as well since, as we will see from Section 4.3, cylindric caustics form before halos. However, soon after they have formed the gravitational instability leads to the growth of a spherical type halo at some point within the cylindric halo. For cylindrical symmetry, the radial density behaves as $\rho \propto r^{-1}$ independent of ϵ_n, and the apoapsis distance power-law q in equation (4.25) scales as $q = (4\epsilon_n - 2)/(9\epsilon_n)$.

4.2.6 Model Eulerian Transformation

As will be shown in Section 4.3 from a simulation, nonlinear evolution leads to an asymptotic phase space profile similar to the self-similar case of the analytical models presented in Section 4.2.5. In this section we consider a form of the Eulerian transformation $\mathbf{X}(\mathbf{q}, t)$ that qualitatively reproduces the behavior of simulations and the self-similar models. We then analyze the analytical model by constructing the caustic surfaces.

Let us consider an already highly evolved halo. As is expected from the analytical models described in Section 4.2.5, the center of a highly evolved halo includes all the particles that were first to fall through the center after the start of the nonlinear phase of the collisionless gravitational collapse. Let us choose a particle close to the center of mass that has the least peculiar velocity with respect to the center of mass of the halo, and let us denote its Lagrangian coordinates by $\mathbf{q_c}$. Having been within the halo for a long time, such a particle will remain extremely close to the center indefinitely.

The phase space structure of the halo where the central particle was selected can be analyzed by plotting Eulerian positions x_i and peculiar velocities v_i of all the particles in Lagrangian planes $q_1 = q_{c1}$, $q_2 = q_{c2}$ or $q_3 = q_{c3}$. For a very evolved halo, its shape is roughly spherically symmetric. Choosing one of the planes $q_1 = q_{c1}$, let us find a generic form of the x_3-component of Eulerian transformation (4.1) for

particles in this Lagrangian plane

$$x_3(q_2, q_3, t) \equiv X_3(q_{c1}, q_2, q_3, t) \ . \tag{4.27}$$

As we know from equation (4.2), as we go to far into the past, Eulerian coordinates approach asymptotically their Lagrangian positions. So, in the zeroth approximation of no perturbation we have equation (4.2) to use in equation (4.27). All the further evolution of structure is only a perturbation around this solution.

Now let us consider the result of the evolution of the halo at the point $\mathbf{q}_c$ where the central halo particle resides. We choose a trial form for the Eulerian transformation

$$x_3 = q_3 - (q_3 - q_{c3})e^{-u^2/b^2} + \alpha \sin\varphi \ , \tag{4.28}$$

where $u^2 \equiv (q_2 - q_{c2})^2 + (q_3 - q_{c3})^2$, b is the perturbation exponential cutoff distance, and α and φ are functions of the position on the (q_2, q_3) plane and are the amplitude and the phase of the oscillatory halo perturbation over the plateau. The second and the third terms in the right hand side of the equation (4.28) are both perturbation terms since they both produce the total displacement of Eulerian mesh with respect to Lagrangian mesh. Let us consider a specific case of

$$\begin{aligned} \alpha &= Au^2/b \\ \varphi &= 2\pi n \left[1 - \exp\left(-\tfrac{u}{\beta b} \ln 2\pi n\right)\right] \ , \end{aligned} \tag{4.29}$$

where A, n and β are positive parameters, so that phase function φ by construction produces no more than n oscillations and therefore maximum of $2n$ caustics in the solution (4.28) (the factor of two is due to the counting of both maxima and minima of the oscillations since they both produce caustics). The parameter β is the ratio of the radial separation u at which the last oscillation occurs to the exponential cutoff distance b.

Consider for example a 400×400 Lagrangian mesh with parameters $b = 150$, $n = 15$, $A = 1.2$, and $\beta = 0.3$. Let the halo have the position $q_{c2}, q_{c3} = (200, 200)$. In Figure 4-3 we present plot of Eulerian transformation given by equations (4.28), and (4.29).

Taking the partial derivative $\partial z_3/\partial x_3$ gives us the Jacobian of the Eulerian transformation or the inverse density. In Figure (4-4) we present the resulting radial density profile. The spikes of density on this plot are at the positions of the surfaces of caustics, or the surfaces where the Jacobian vanishes.

4.2.7 Caustic Destruction and Particle Resolution Limitations

Here we describe the effects leading to the destruction of caustics. As we find below, these effects are significant only when the number of particles in a halo is small, making all of the destruction effects important only for some simulation parameters, but not for the real neutralinos in the universe.

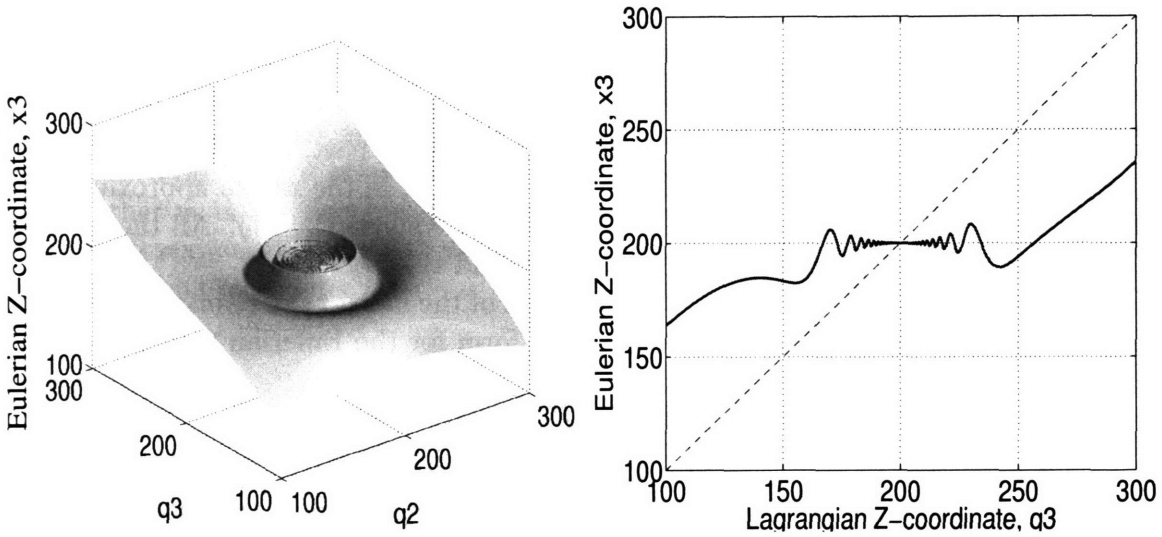

Figure 4-3: The Eulerian transformation in the vicinity of the halo at the point in plane $(q_2, q_3) = (200, 200)$, using equation (4.27) and (4.29), with parameters defined in the main text. Left: The diagonal view. Right: The cross section of the q_3 dimension (solid line), compared with the unperturbed position (dashed line).

Limits by numerical particle resolution

The caustics will manifest their properties in a simulation only when their phase space structure is well resolved with the simulation particles. Further in this section we will find that the spherical and cylindrical fold singularities are observed in an N-body simulation.

Consider a set of enclosed spherical caustic surfaces formed by cold dark matter particles. Labeling the shells by an index i, the number of particles contained in a caustic by N^i and its radius by R_i, and assuming there are already many enclosed spherical shells, we can consider their local radial separations $\Delta R_i \equiv R_i - R_{i-1}$ to be locally uniform $\Delta R_i \approx \Delta R_{i+1}$, also $N_i \approx N_{i+1}$. The condition on the mean particle separation on the caustic surface required in order to distinguish the caustic i in physical space from the adjacent caustics $i-1$ and $i+1$ is that the mean interparticle separation be less than the radial separation ΔR of the adjacent caustics, or

$$n_{\rm res} \equiv \frac{N}{4\pi} \left(\frac{\Delta R}{R} \right)^2 \gg 1 \,, \tag{4.30}$$

where we suppressed the spherical shell index i, and the first multiplier in the right-hand side is the surface density of particles, valid for the case of spherical symmetry. A similar constraint can be written for cylindrical and planar caustics.

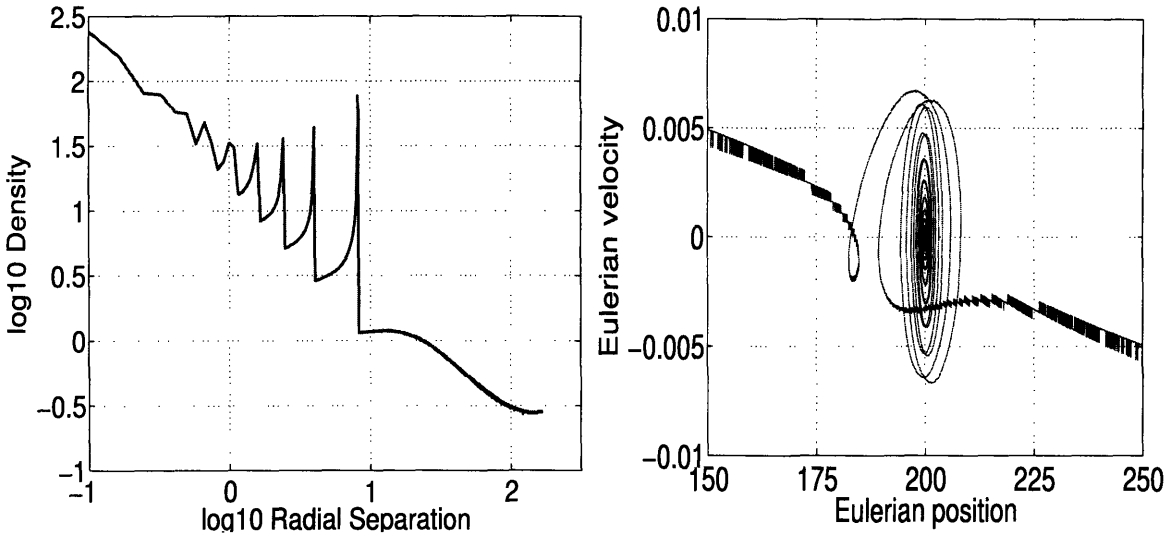

Figure 4-4: Left: Caustics are manifested as the density singularities. Shown here is the radial density distribution for the halo in Figure 4-3. Right: The phase space portrait of the whole system at a given time. The curve is unsmooth due to numerical finite difference approximation errors.

Caustic destruction by two-body relaxation

Consider a particle flying through a cloud of other particles, as depicted by Figure 4-5. Treating the trajectories of a pair of particles as straight lines, the component of the force in the perpendicular direction is $F = F_\perp \cos\theta$, where F is the force acting between the particles. Using the Plummer force law acting between the simulation particles, we have

$$F_\perp = \frac{Gm^2 r}{(r^2+\epsilon^2)^{3/2}} \cos\theta = \frac{Gm^2 b}{(x^2+b^2+\epsilon^2)^{3/2}} .$$

By Newton's second law, $\dot{\mathbf{v}}_\perp = \mathbf{F}_\perp/m$. Using $x = vt$, we get

$$\delta v_\perp = \int \left(\frac{F_\perp}{m}\right) dt = \frac{2Gmb}{(b^2+\epsilon^2)v} , \tag{4.31}$$

which is the contribution due to the gravitational interaction with a particle at the impact parameter b. When the particle is moving with the velocity v through a cloud of other particles of number density n, the number of particles encountered at the time dt within the range of the impact parameter b to $b+db$ will be $nv\,2\pi b db\,dt$. This makes the contribution to the mean squared perpendicular velocity due to all the surrounding particles

$$\Delta v_\perp^2 = \int \left(\frac{2Gmb}{(b^2+\epsilon^2)v}\right)^2 2\pi nvb\; db\, dt = \frac{4\pi n G^2 m^2}{v} I\; dt , \tag{4.32}$$

where

$$I \equiv \int_{b_{\rm min}}^{b_{\rm max}} \frac{b^2\, db^2}{(b^2+\epsilon^2)^2} = \left[\frac{\epsilon^2}{b^2+\epsilon^2} + \ln(b^2+\epsilon^2)\right]\Bigg|_{b_{\rm min}}^{b_{\rm max}} \tag{4.33}$$

is the Coulomb integral. The inverse square interparticle force law corresponds to $\epsilon = 0$.

The parameter $b_{\rm max}$ in equation (4.33) is the maximum effective impact parameter, equal to the correlation length of particle density or approximately the size R of a halo $b_{\rm max} \sim R$. The parameter $b_{\rm min}$ is the impact parameter below which the approximation of the trajectory of particles with a straight line is violated, or equivalently below

$$v^2/2 \sim Gm/(b_{\rm min}^2+\epsilon^2)^{1/2}\,. \tag{4.34}$$

The parameter $b_{\rm min}$ equals zero when the straight line approximation is valid at any impact parameter which happens when the smoothing length is so high that the particle trajectories are not deflected from the straight lines, implying $v^2/2 > Gm/\epsilon$, or

$$\epsilon > R/N_c\,, \tag{4.35}$$

where we used

$$v^2/2 \sim GmN_c/R\,, \tag{4.36}$$

where N_c is the number of particles in a relaxed halo where the motion is considered. Equation (4.35) is usually satisfied in an N-body simulation. Indeed, expressed in code units (tildes: the units of PM force grid spacing), that condition reads $\tilde{\epsilon} > \tilde{R}/N_c \approx 1/(p\tilde{R}^2)$, where p is the number of particles per PM-grid spacing in a halo. The typical values are $p \gg 1$ and $\tilde{R} \gg 1$ for a halo, while the reasonable $\tilde{\epsilon}$ value, based on the force accuracy, as discussed in Chapter 3, is $\tilde{\epsilon} \sim 0.1$. Neglecting the first term in equation (4.33) because it is always less than or equal to unity the Coulomb integral is

$$I = 2\ln(R/\epsilon), \quad \epsilon > R/N_c\,. \tag{4.37}$$

If condition (4.35) is not satisfied, the parameter $b_{\rm min}$ is found from equations (4.36), and (4.34). The resulting Coulomb integral (4.33) is

$$I = 2\ln(N_c), \quad \epsilon < R/N_c\,. \tag{4.38}$$

The two above equations combine, giving

$$I = 2\ln\left[\min\left(N_c, \frac{R}{\epsilon}\right)\right] = \ln\Lambda, \quad \text{where} \quad \Lambda \approx \frac{N_c^2R^2}{\epsilon^2N_c^2+R^2}\,. \tag{4.39}$$

The collisional relaxation time, defined as the time when the correction of the velocity squared is comparable to the velocity itself $\Delta v_\perp^2 \approx v^2$

$$t_{\rm relax} = \frac{v^3}{4\pi\rho G^2 m I}\,, \tag{4.40}$$

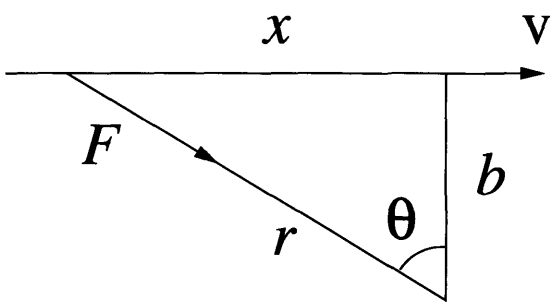

Figure 4-5: A test particle passes by another particle at the impact parameter b with velocity v.

where $\rho = nm$ is constant independent of particle discretization.

For the real neutralinos filling the universe, the relaxation time given by this formula is always much greater than the Hubble time, as illustrated by an example from the solar neighborhood. Using the values $v = 350$ km s^{-1}, $m = 100$ GeV, and $\rho = 0.3$ GeV c^{-2} cm^{-3} as a rough estimate of the local density of dark matter [27], and $I = 100$, we have $H_0 t_{\rm relax} \approx 2 \times 10^{62}$. Two body relaxation is unimportant for neutralinos.

In an N-body simulation the relaxation time becomes significantly smaller due to the coarse sampling of the mass with simulation particles. The mass of a particle in a simulation box of comoving size L in a simulation with a box of N particles is

$$m_{\rm sym} = \frac{3H_0^2\Omega_m L^3}{8\pi G N} \approx 4 \times 10^{40} \left(\frac{L}{\rm pc}\right)^3 \left(\frac{1024^3}{N}\right) \text{ GeV} . \tag{4.41}$$

For a virialized clump of matter of size R, we have equation (4.36). Using this in equation (4.40), we get

$$t_{\rm relax} = \frac{N_{\rm c}}{4\pi I}\, t_{\rm dyn} , \tag{4.42}$$

where $N_{\rm c}$ is the total number of particles in the dynamically evolved clump of minimal size, and $t_{\rm dyn}$ is the dynamical time within the virialized clump, proportional to $1/\sqrt{G\rho}$ where ρ is the density within the virialized clump. This equation shows that the relaxation time in a dense clump grows linearly with the number of particles in the clump.

As we noted above, the matter distribution is described as a fluid when the interparticle distance is much less than the characteristic distance at which the velocity field changes (the size of the virialized clump), which is equivalent to increasing $N_{\rm c} \to \infty$ in equation (4.42). One way of reaching a given minimal number $N_{\rm c}$ of particles in each of the matter perturbations in the simulation box is to cut off the initial matter power spectrum above the wave number $k_{\rm c}$ such that volume of size $l_{\rm c} \equiv 2\pi/k_{\rm c}$ contains the desired number of particles. This introduces a coherence scale $l_{\rm c}$.

The dynamical time $t_{\rm dyn}$ in equation (4.42) is approximately equal to the turnaround

time for spherically symmetric caustics with the same amplitude of initial density perturbation. In [5], the initial turnaround time $t_{\rm ita}$ is given as a function of the initial spherical overdensity δ_i at the initial time t_i by

$$t_{\rm ita} = \frac{3\pi}{4}\delta_i^{-3/2} t_i \ . \tag{4.43}$$

If our simulation is starting at a given expansion factor a_i with matter perturbations of a given δ_i, the expansion factor $a_{\rm ita}$ at which first caustic turnaround occurs is given by

$$\frac{a_{\rm ita}}{a_i} = \left(\frac{t_{\rm ita}}{t_i}\right)^{2/3} = \left(\frac{3\pi}{4}\right)^{2/3} \delta_i^{-1} \tag{4.44}$$

For a plane parallel caustic, as opposed to the spherical one, the coefficient in parentheses in the right-hand side of the above equation is unity.

Combining equations (4.43) and (4.42), we find that the effect of two-body relaxation is negligible throughout the simulation only when it is evolved up to the expansion factor

$$a < a_{\rm relax} \equiv a_i \left(\frac{3N_{\rm c}}{16I}\right)^{2/3} \sigma_i^{-1} \tag{4.45}$$

where N_c is the number of simulation particles in the volume of sidelength equal to the power spectrum cutoff scale $l_{\rm c}$, $N_c \approx N(l_{\rm c}/L)^3$, and $\sigma_i = \langle \delta_i^2 \rangle^{1/2}$ is the initial rms amplitude of fluctuations set in the simulation.

In a typical cosmological N-body simulation, the small scale power spectrum cutoff is not applied, the first clumps of particles to form have few particles in total, $N_c \approx 1$. Equation (4.45) yields immediately that the two-body relaxation completely disperses all the substructure of these clumps within a few clump dynamical times. In order to observe caustics, we require $N_c \gg 1$, or power spectrum cutoff scale $k_{\rm c} = 2\pi/l_{\rm c} < 2\pi L/N$ in our simulations. Caustics can survive in such simulations for many dynamical (shell crossing) times.

Temperature Effect on Caustics, and Heating by Relaxation

The analysis in Section 4.2.4 is modified when the matter is no longer cold. Finite temperature (random velocity of particles) leads to smoothing of the caustics.

For the fold caustics, the matter distribution can be described as locally plane-parallel. For the evolved plane parallel singularity of finite temperature, the peak density is given by [17] as

$$\rho_{\rm max} \approx 2\rho_0 \kappa^{-1/2} \left\langle (v_{\rm th}/\sigma_{\rm th})^{-1/2} \right\rangle \tag{4.46}$$

where $\sigma_{\rm th}$ is the rms value of the thermal velocity $v_{\rm th}$, $\kappa = \sigma_{\rm th} t_1 l_1^{-1}$, where t_1 and l_1 are the proper time and length scale of the caustic at its formation.

The characteristic width of the caustic density spike $l_{\rm th}$ is given by equating the surface density of the caustic $\rho_{\rm max} l_{\rm th}$ with the integrated column density of the perfect

fluid caustic, whose density profile is given by

$$\rho(t_1, x) = 2\rho_0 (x/l_1)^{-1/2} , \tag{4.47}$$

where ρ_0 is the average background density. Following this procedure, we find

$$l_{\rm th} = \frac{4 l_1 \kappa}{\left\langle (v_{\rm th}/\sigma_{\rm th})^{-1/2} \right\rangle^2} \tag{4.48}$$

It is important to analyze the processes that can lead to the thermal heating of CDM, because in the presence of a finite particle velocity dispersion, the thickness of caustics may exceed their spatial separation, making them indistinguishable in the simulation.

Let us consider a particle with velocity v. After particles turn around, streams of cold dark matter with opposite bulk velocities pass through each other. Particles in those streams thermalize with the characteristic time $2t_{\rm relax}$, where an additional factor of two enters due to the opposite velocities of the streams, reducing equation (4.32). The heating rate per particle is given by

$$\Gamma \approx \frac{mv^2}{4t_{\rm relax}} \approx \left(\frac{Gm^2}{vt_{\rm dyn}}\right)\left(\frac{I}{t_{\rm dyn}}\right) . \tag{4.49}$$

Let us make a crude estimate of the time in the simulation when the caustics become indistinguishable as a result of heating. Using equation (4.32), we find that at time t_1, the thermal velocity dispersion is approximately

$$\sigma_{\rm th}^2 = \frac{2\pi\rho G^2 m I}{v} t_1 . \tag{4.50}$$

Plugging this into equation (4.48) and using $v \approx l_1/t_{\rm dyn}$ and $t_{\rm dyn} \approx (G\rho)^{-1/2}$ we find

$$l_{\rm th} = \frac{2 l_1}{\left\langle (v_{\rm th}/\sigma_{\rm th})^{-1/2} \right\rangle^2} \left(\frac{4\pi I}{N_{\rm c}}\right)\left(\frac{t_1}{t_{\rm dyn}}\right)^{3/2} . \tag{4.51}$$

Suppose that the separation between the caustics of interest is Δl_1. The caustics are distinguishable from each other up until thermal heating thickness exceeds their separation $l_{\rm th} < \Delta l_1$, or

$$t < t_{\rm dyn} \left(\frac{N_c}{8\pi I}\frac{\Delta l}{l_1}\right)^{2/3} \left\langle (v_{\rm th}/\sigma_{\rm th})^{-1/2} \right\rangle^{4/9} . \tag{4.52}$$

We see that heating is important for small clumps but not as $N_{\rm c} \to \infty$.

Effect of Integrator Errors

Numerical integrator errors lead to a similar effect on caustics as heating by two-body relaxation.

The integration errors depend on the integration scheme and increase as the number of integrator timesteps increases. In order to put a limitation on a maximum integration error within a box, the trajectory of each of the particles should be sampled by a large number of timesteps each local dynamical time. Choosing the same timestep for each particle in the simulation volume equal to a fraction of the minimum local dynamical time within the whole simulation will automatically satisfy this constraint. The minimum dynamical time is achieved in systems of the highest density. For pointlike particles, such a density can in principle reach infinite value. By using the Plummer force softening length ϵ we effectively consider the particles as ϵ-sized spheres, in which case the minimum dynamical time is achieved for high density systems where density is averaged over the range of force softening length. For such a system, the dynamical time is given by

$$t_{\rm dyn}^{\rm min} = \frac{\pi}{2}\sqrt{\frac{\epsilon}{g}}\,, \tag{4.53}$$

where g is the gravitational acceleration caused by the particles within the smoothing length. The parameter η_t in our time step selection scheme

$$\Delta t = \sqrt{\frac{\epsilon\eta_t}{g_{\rm max}}} \tag{4.54}$$

defines the fineness of sampling of the orbit around the densest spot in the simulation volume.

In the densest point of the simulation volume, the dynamical time is minimal. At such point, the test particle orbit around the center of the density perturbation will be sampled by at least

$$N_{\rm orb}^{\rm min} = \frac{T_{\rm orb}}{dt} = \frac{4t_{\rm dyn}}{dt} = \frac{2\pi}{\sqrt{\eta_t}} \quad \text{timesteps.} \tag{4.55}$$

For example, for $\eta_t = 0.05$, this gives us 28 points along one rotation along the orbit.

In phase space, integrator errors lead to deviations around the perfect trajectory which are limited in the extent (by KAM-theorem) due to the symplectic nature of the integrator chosen. Due to the errors of integration, the caustic surface, being perfectly shaped in the exact solution, will have a finite thickness in the numerical simulation. This effect of caustic destruction is expected to be relatively small since the orbital phase changes tend to be coherent for symplectic integrators.

4.3 Caustic N-body Simulations

The self-similar models discussed in [25] and [5] are completely solved analytically and are idealizations to the cases of high (cylindrical, spherical, planar) symmetry of the initial conditions. The self similarity of the solutions itself implies infinite revolutions of matter around the center of coordinates in phase space, which is an additional

idealization, since in the initial conditions matter starts from rest in comoving coordinates. The analytical models provide precise solutions for the times before the first spherical matter infall and for the self similar state of matter distribution to which a system reaches at late times. In addition, the analytical models treat only a special case of the initial density perturbations (top-hat in [5] and power-law in [25]).

Simulations allow us to study the generalization to matter distributions that lack the assumptions of any symmetry in the initial conditions, and in addition to study the evolution of the system in the intermediate state where matter has not yet reached its asymptotic state. Also, simulations allow us to consider a hierarchy of collapsed halos through the sampling of the primordial power spectrum of density perturbations.

Studying the statistics of the collapsed halos requires an extremely high particle number in the simulation box, since it requires having a number of evolving halos, each of which being sampled with enough particles to resolve their caustic substructure so that numerical two-body relaxation is negligible.

The only N-body simulation of dark matter particles on the scales of the interstellar distance known to us where the appearance of caustics is close to plausible is [16], who used the same initial conditions as we do, but they did not sample the smoothing scale of the initial conditions with enough particles. Caustics did not appear in their simulation, because their initial condition smoothing scale contained only ≈ 40 particles. Therefore, due to excessive two-body relaxation the caustics were destroyed before the formation of the first nonlinear structures. In the simulations presented below, the smoothing scales in the initial conditions contain many more particles which rendered the appearance of caustics in the simulation and their survival into the late nonlinear stage possible.

4.3.1 PM simulation of caustics

Soon after our PM-parallel code was developed, we ran a PM-simulation of caustic evolution with 512^3 particles and 512^3 PM-density mesh. The PM-code, with an S_2-sphere parameter $\tilde{\eta} = 3.3$, is roughly equivalent to the P^{3}M simulation with Plummer force softening length

$$\tilde{\epsilon} \approx 0.33\tilde{\eta} \approx 1.1 \; . \tag{4.56}$$

Recall that in a typical P^{3}M simulation $\tilde{\epsilon} \ll 1$, allowing sub PM-grid resolution. A PM code does not allow sub-grid force resolution.

The smoothing length parameter l_s in the initial density power spectrum

$$P(k) = P_0 k^n \left[\frac{D_+(a)}{D_+(1)}\right] \exp(-k^2 l_s^2) \; , \tag{4.57}$$

is chosen so large that the effect from unresolved scales on structure formation is negligible. Setting $l_s = L/4$, the whole simulation volume L^3 contains only $(L/(\pi l_s))^3 = (4/\pi)^3 \approx 2$ initial density peaks. We chose a minimal power spectrum slope $n = 0$ for initial conditions however at the caustic scales.

In Figure 4-6 we present the projected density distribution in a slab of our volume at a timestep 840 and the expansion factor $a \approx 20 a_i$, where a_i is the initial expansion

factor in the linear regime when $\sigma_i \approx 0.2$. The shallow density perturbations which are imprinted into the initial conditions have contracted and occupy the compact volumes, starting the formation of two major halos.

The lower halo in the figure at this time already has developed an extensive substructure, while the upper halo is still at the very initial phase of formation. Those halos represent significant density enhancements, and their structure is very roughly described by spherical.

Each of the two major halos consists of many shells each enclosed into one another like an onion structure. It is visible already from the physical scales shown in Figure 4-6 that density distribution has jumps in both the spherical and cylindrical structures, which is in qualitative agreement with the prediction of the analytical model of caustics. Within the space between these halos, matter falls towards one or the other of them. In addition, matter from the surrounding medium close to the line connecting the halos in the middle between them falls towards this line under gravity, forming a roughly cylindrical filament.

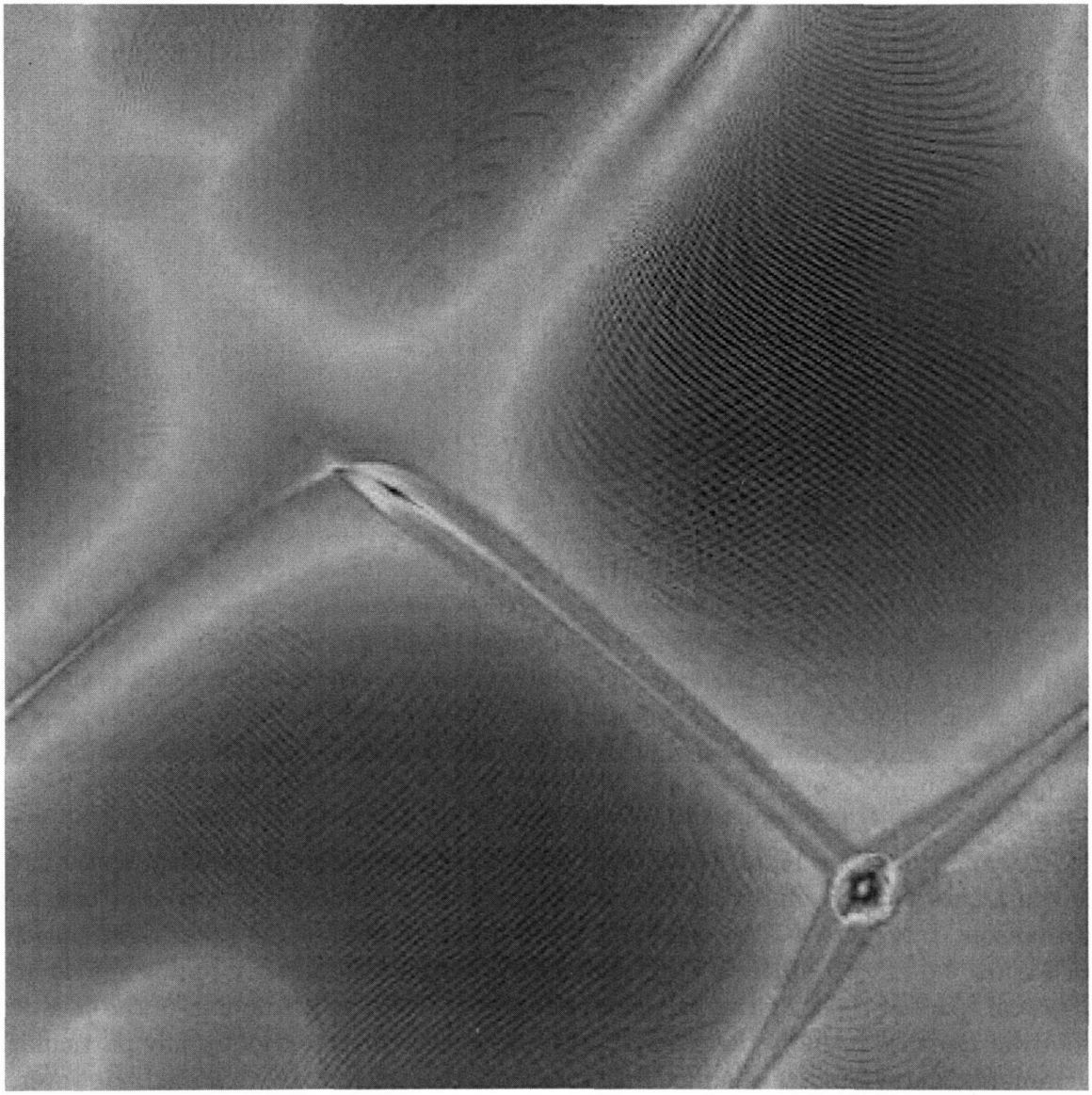

Figure 4-6: Projected physical space density distribution in the PM-simulation timestep 840 and the expansion factor $a/a_i \approx 20$. The image spans the whole simulation box. The density is projected from a slab of the thickness equal to a quarter of the whole simulation volume. Two major halos are forming, one of them already being collapsed, and the other at the initial stage of formation. The filament between the halos has roughly cylindrical structure. The lower halo is roughly spherical, the upper one is at the initial stage of formation and therefore did not yet evolve any symmetry. The arch-like structure in the bottom left in the voids are due to matter distributed in a structure that does not completely fit the slab, and are not due to caustics.

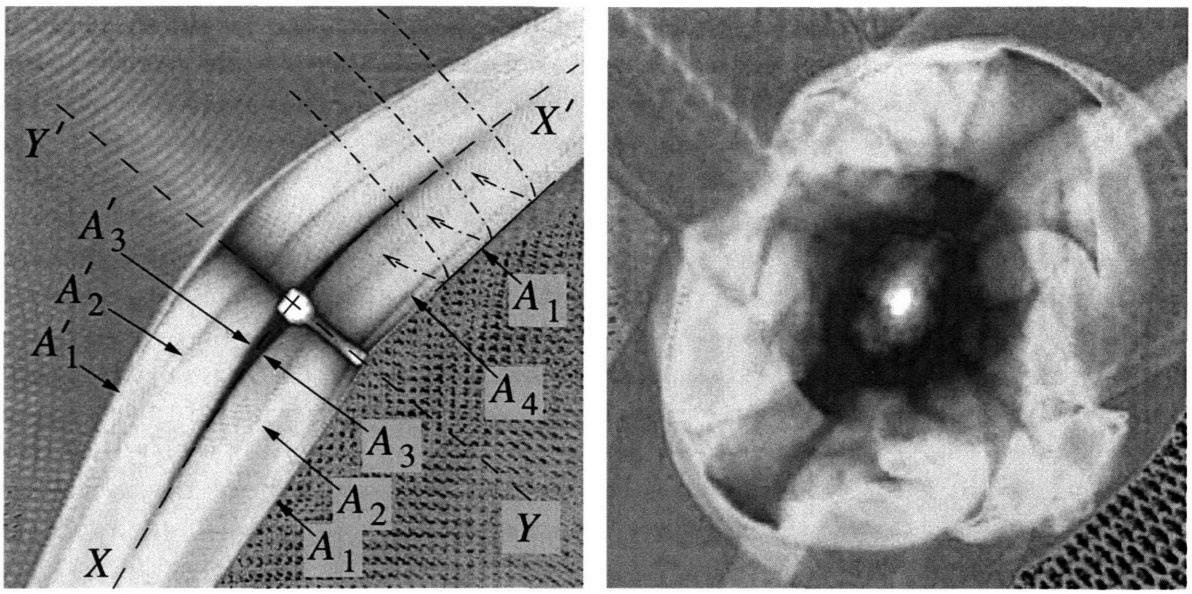

Figure 4-7: The first nonlinear structure to form in the simulation box. The density distribution is shown within the 30×30 PM-density grid spacing volume of the halo in the bottom-right corner in Figure 4-6. Left: The snapshot at an early stage of formation, just after the first shell crossing of the center of coordinates. The typical trajectories followed by particles in the volume are shown with dash-dotted lines. The envelope curve of the family of the trajectory paths form the caustic surface. Right: A snapshot after about 10 crossing times. At this time a quasi-spherical halo has formed.

Zooming into the halos at different timesteps shows that in physical space the substructure is very similar to that expected from the analytical model of caustic formation. In Figure 4-7 we present a zoom-in of the halo in the lower right corner of Figure 4-6 at two different stages of the evolution. The density jumps appear in physical space in those figures precisely as expected from the predictions of the caustic models. Because initially the matter in the simulation does not follow any particular symmetry, the symmetry assumed for those models is not precisely followed. However, gravity tends to make the structures become more symmetrical over time.

In the left panel of Figure 4-7 we observe the early development of the first complicated structure to form in the whole simulation box. The caustic fold singularities are visible on this figure as the smooth surfaces of high density contrast. The first structures formed have cylindrical symmetry with the axis XX' rather than spherical symmetry. Qualitatively, the scheme outlined in [25] and [5] is very well followed. The matter that has passed through the center of symmetry expands reaching the maximum radial separation. Then it turns around and falls back again toward the central halo. Then the cycle repeats. The caustics or the smooth surfaces of high density contrast are the envelope curves to the paths of the particles on the plane of the image.

By the time shown, there are more than four cylindrical caustics already formed (the pairs of folds (A_1, A_1'), (A_2, A_2'), (A_3, A_3') and the fold A_4, which does not have a

counterpart in this projection), corresponding to roughly four turnaround times of a matter in cylindrical caustics with the common axis of symmetry XX'. The topology of a caustic surface in three dimensions is that of the two dimensional wrapped sheet. Over the time, the axis of the cylindrical quasi-symmetry bends due to the gravitational influence of the matter as a surrounding medium. The cylindrical symmetry is completely broken within the inner portions of the halo, where a quasi-spherical halo is forming.

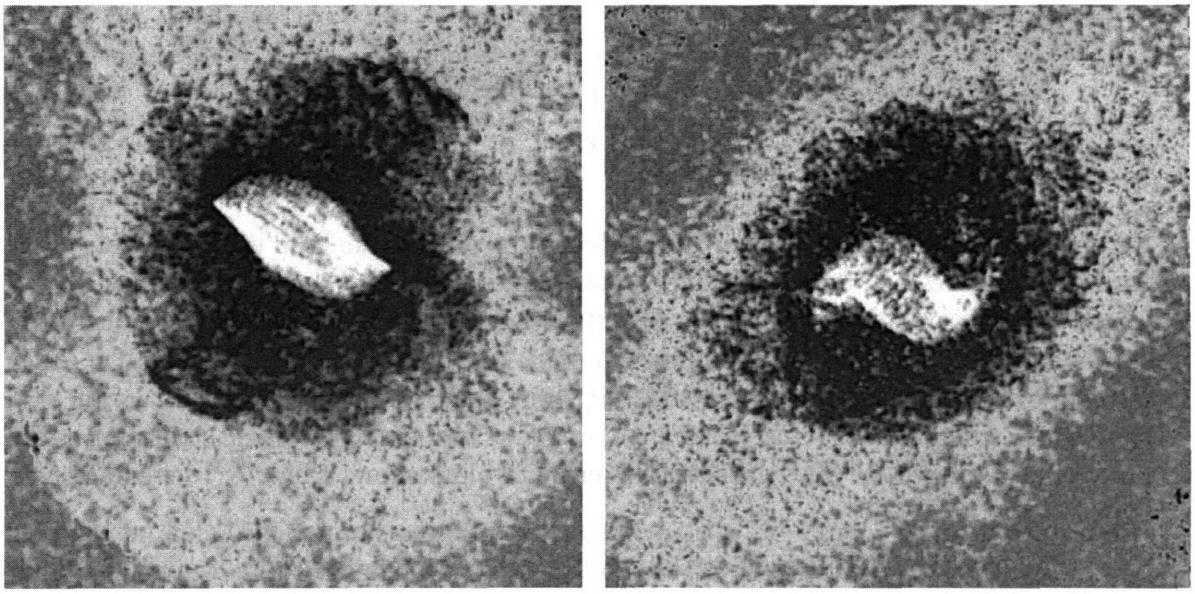

Figure 4-8: Innermost caustic. Left: The density distribution within a volume of size 3 PM-density mesh spacings containing the inner caustic. Left: After 6 central revolutions (dynamical times). Right: Snapshot at the 15th revolution.

A quasi-spherical halo forms in the center as a result of the collision of the two ends of the quasi-cylindrical structure (the left panel in Figure 4-7) into each other under the influence of gravity. The central halo forms first as a small central singularity formed by the collapse of matter through the center of the structure. The structure continues to evolve faster than the rest of the matter anywhere in the simulation box. The turnaround radii of the caustics expand, the outermost fold caustic surface serving as a boundary between the halo and the surrounding medium. The innermost caustic structure evolves as each revolution of matter generates a new expanding shell. All the shells are part of a two dimensional sheet in the three dimensional physical space, just like in the analytical models of [25] and [5]. It is striking that the inner caustic structure survives very many revolutions. We have followed the structure up to 30 revolutions and later at which point the effects of numerical two-body relaxation considered in Section 4.2.7 still have not affected the caustics. Although, the small scale structures have shrunk in physical space, the size of the innermost caustic is sufficient to contain many particles even at late times.

Radial density profiles can be used to test the self-similarity of the caustic structure. In Figure 4-9 we present the spherically averaged radial profile for the large

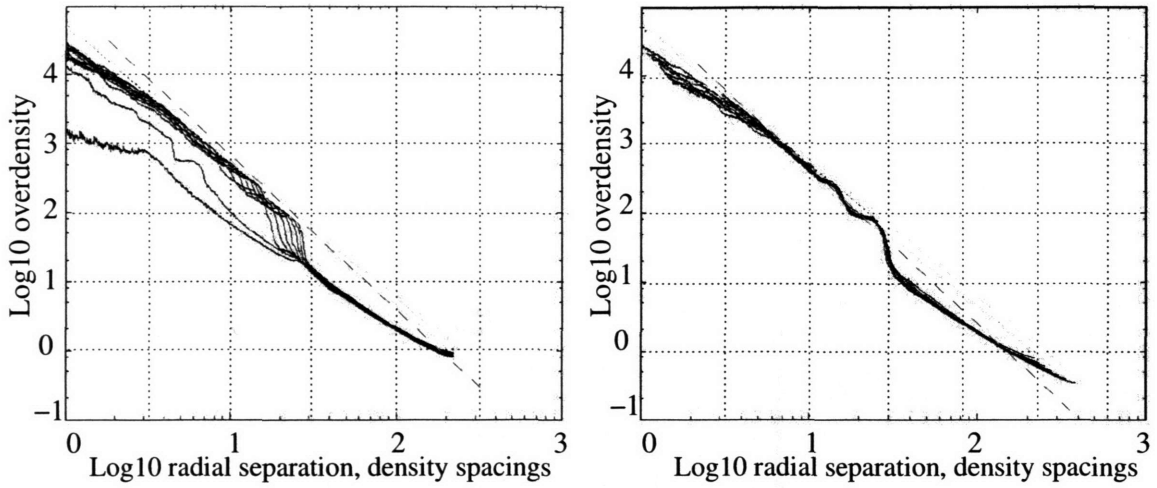

Figure 4-9: Spherically averaged radial density profile of the halo in the bottom right corner in Figure 4-6. Left: Density versus comoving radius for several times during which the universe doubles in size. Right: Testing the self-similarity by shifting the curves in the radial and vertical directions to minimize their separation. The lower two curves on the left panel are omitted since they show the earliest stage of halo formation, when the structure formation was far from the asymptotic structure in the limit of self-similarity. The dashed line on both plots shows the $r^{-9/4}$ dependence.

halo. The caustics, manifested as the infinite spikes in the density plots in the models in [25] and [5] disappear here as a result of the angle averaging in our non-spherically symmetric halo. The caustics instead show up as wiggles above the power-law. The self-similarity of the solution is tested by shifting the curves in the horizontal and vertical directions to minimize their separation and observing whether their profiles coincide in the result. As we see in the right panel in Figure 4-9, this procedure indeed results in very similar profiles, confirming the approximate self-similarity of the models. This figure shows good qualitative agreement with the results [5], as expected in the model with power spectrum slope $n = 0$ and $\epsilon_n = 1$ in Section 4.2.5.

The most interesting result from the simulation is the phase space structure of halos. In Figure 4-10, we present the phase space diagram of the whole matter distribution as a scatter plot in the $(\ln r, v_r)$ plane. The left panel shows the phase space portrait of the structure as it starts to evolve into the nonlinear regime, at which point only a few orbits have occurred. In the right panel, the matter is highly evolved. One can count about 14 rings in the structure, corresponding to 14 orbits through the center. In physical space we observed exactly the same number of crossings of the inner caustic. The inner caustic serves as a generator of the phase space structure. The inner caustic collapses periodically, and each new curl formed by this recollapse propagates into the surrounding portions of the halo. This quite impressive illustration of the phase space was possible only due to the near spherical symmetry of this halo. The structure shown in this figure is a projection from six dimensional phase space.

In both the real universe and the simulation, the caustics form differently from

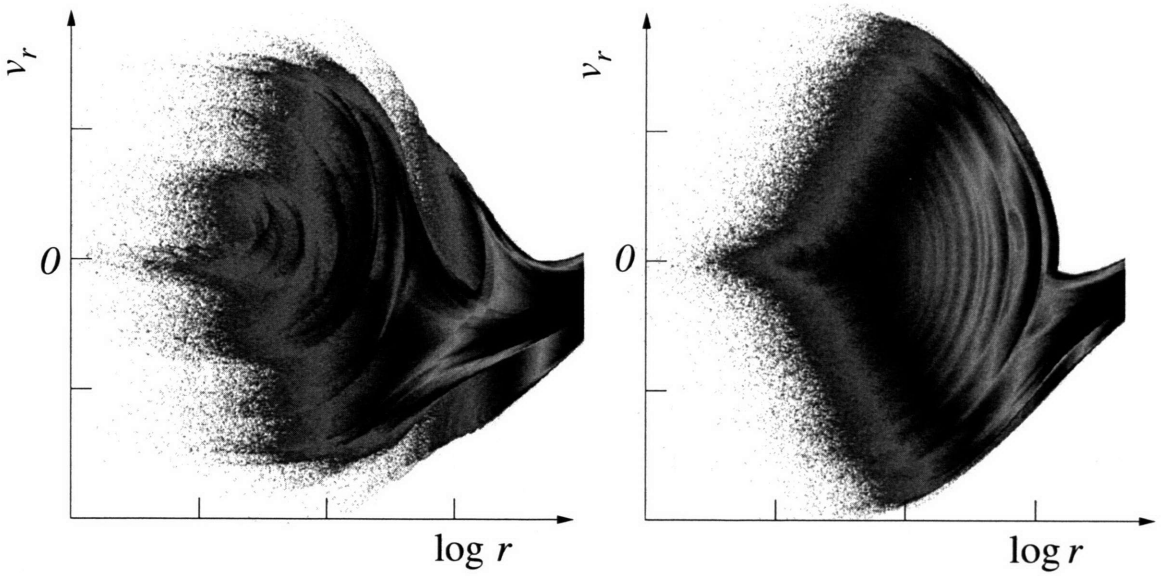

Figure 4-10: The projection of matter distribution onto the $(\ln r, v_r)$ - plane, where r is the proper distance from the major halo shown in Figure 4-6 and v_r is the proper peculiar radial velocity. The left and right panels show the distribution at expansion factors $a/a_i = 16$ and $a/a_i = 20$ respectively.

analytic models that assume symmetry during each of the stages of formation. Nevertheless, as our simulation shows, the caustics do form and are long lived.

4.3.2 P^{3}M Caustic Simulation and the Initial Conditions

The PM-simulation described in the previous section shows the survival of the inner caustic. However the dynamic range of the structure formed is constrained by the initial conditions allowing only two waves for the whole simulation volume, which excluded such events as mergers or a hierarchy of structure formation. Our simulations are designed to test the self-similar models of Section 4.2.5. Note however that simulations suffer from the limitations of discrete power spectrum sampling, finite dynamic range, periodic boundary conditions, deviations from exact spherical symmetry, and finite mass resolution.

In this section we describe a P^{3}M simulation performed in addition to the PM simulation described in previous section. There are three differences between them. Compared with the PM simulation, the P^{3}M simulation we describe below
1) has a factor 262 times fewer particles per initial coherence volume
2) has smaller force softening length ($\tilde{\epsilon} = 0.1$ instead of 1.1)
3) has a different slope of the large scale power spectrum ($n = -3$, instead of $n = 0$).

The simulation parameters (the smoothing length and the redshift of the epoch of first nonlinearity) in the initial conditions were tied to replicate the results in [29].

We assumed $\Omega_m = 1$ for our caustic N-body simulation. The PM-force resolution parameter η was set to 3.3 PM density grid spacings. The P^{3}M force resolution parameter ϵ was set to 0.1 PM density grid spacings. We used an 800^3 grid.

The perturbations were generated using equation (4.57) with $n = -3$. This power spectrum includes the damping with $k_c \approx 1.77 \times 10^6$ Mpc^{-1}, due to Silk damping at neutralino kinetic decoupling [29]. The damping scale wave number $k_c \equiv 1/l_s$ where l_s is our smoothing length parameter. At k_c the power is transforming from the nonlinear large scale cluster evolution spectrum $P(k) \propto k^{-3}$ for $k < k_c$ to the power spectrum extinguished by free streaming and collisional damping. Our model power spectrum is designed to replicate the initial power spectrum at $z \approx 350$ leading to the epoch of nonlinearity at $z \approx 60$. We use equation (4.57) for the power spectrum instead of the form used by [29] because the latter does not have an accurate treatment of free-streaming damping.

The simulation box PM gridsize N and the sidelength L should be chosen so that the smoothed halos of size $(\pi l_s)^3$ include a large number of particles, so that the constraint (4.45) should be well satisfied. On the other hand, we want to study the hierarchical structure formed within the caustics, and therefore to observe structure formation over a range of scales. These competing desires require that the grid size of the simulation is as large as possible. With the scalable and load balanced N-body code presented in the previous chapters it is possible to simulate the particles on a large grid far into the nonlinear regime. As a practical compromise, we choose $N = 800$ and $l_s \equiv L/n_{\rm smoo} = L/40$. This gives a factor of 10 increase in the dynamic range of length scales compared with the PM-simulation. At the same time, each initial perturbation is sampled with a factor of 262 times fewer particles.

The normalization of the power spectrum P_0 is chosen so that the growing mode becomes nonlinear at redshift $z_{\rm nl} \approx 60$. The Zel'dovich approximation (4.14) is accurate while the rms density fluctuation $\sigma^2 \equiv \langle \delta^2 \rangle < 0.2$. We normalized the power spectrum so that

$$\left(\frac{a\sigma_i}{a_i} \right)^2 = 1 \quad \text{at} \quad 1 + z = a^{-1} = 61 \, .$$

We set $\Omega_m = 1$ and neglected a cosmological constant because the latter is irrelevant at $z \gg 20$.

4.3.3 Analysis of the P^{3}M Simulation

The simulation started at a redshift $z_i = 350$ and evolved up to the final redshift $z = 50.2$, at which point the structure has become nonlinear on the scale of the box. Throughout the following discussion we will measure comoving distances in code units (PM grid density spacings). One grid spacing equals $\Delta x = 0.0282$ comoving pc. Our present Hubble constant is $H_0 = 71$ km s^{-1} Mpc^{-1}, and one simulation particle weighs 3.13×10^{-12} M$_\odot$.

The random density perturbations in the initial conditions had rms $\sigma = 0.196$, while the maximum was $\delta = 0.787$. According to the spherical model of density perturbations, a perturbation collapses when the overdensity predicted on the basis of linear theory is 1.69. Since in the CDM model of structure formation linear perturbations grow with $\delta \propto a$, a typical perturbation should collapse when the expansion factor reaches the ratio $1.69/0.196 \approx 8.9$ times its initial value, or at red-

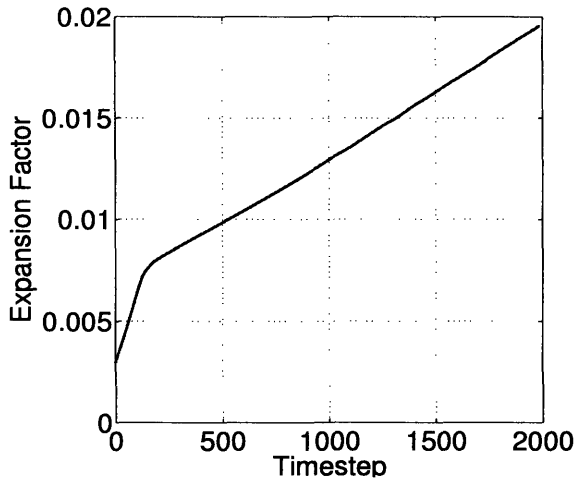

Figure 4-11: Expansion factor $a = (1+z)^{-1}$ as a function of the timestep in the P^{3}M caustic simulation.

shift $350/8.9 \approx 39$. On the other hand, the first perturbation should collapse at redshift $350 \times (0.787/1.69) = 162$, which occurs after 100 timesteps. In fact, by this redshift we indeed see the first collapsed structures form in our simulation box. In Figure 4-12 we present images of the density distribution from the initial conditions and later when the first caustic forms within the simulation volume.

At the later timesteps this perturbation (*halo I*) evolves and others develop. Halo I remains the most apparent one, undertaking many mergers across the whole simulation. In Figure 4-13 we present the whole simulation volume at last timestep 1983 and in Figure 4-14 we show the halo and its surroundings (in different projection) as they were at redshift 50.2. The halo shows extremely complicated caustic structure. The halo is under the continuous process of merging with the other halos which themselves have an evolving caustic structure. The outer caustic structure shown in Figure 4-14 is more complicated than the structure of any of the halos in the PM-simulation of Section 4.3.1. Because our P^{3}M initial conditions allowed more perturbations (waves) in the initial conditions those waves developed and provided an extremely convoluted structure in phase space. When projected to physical space this structure results in this very complicated set.

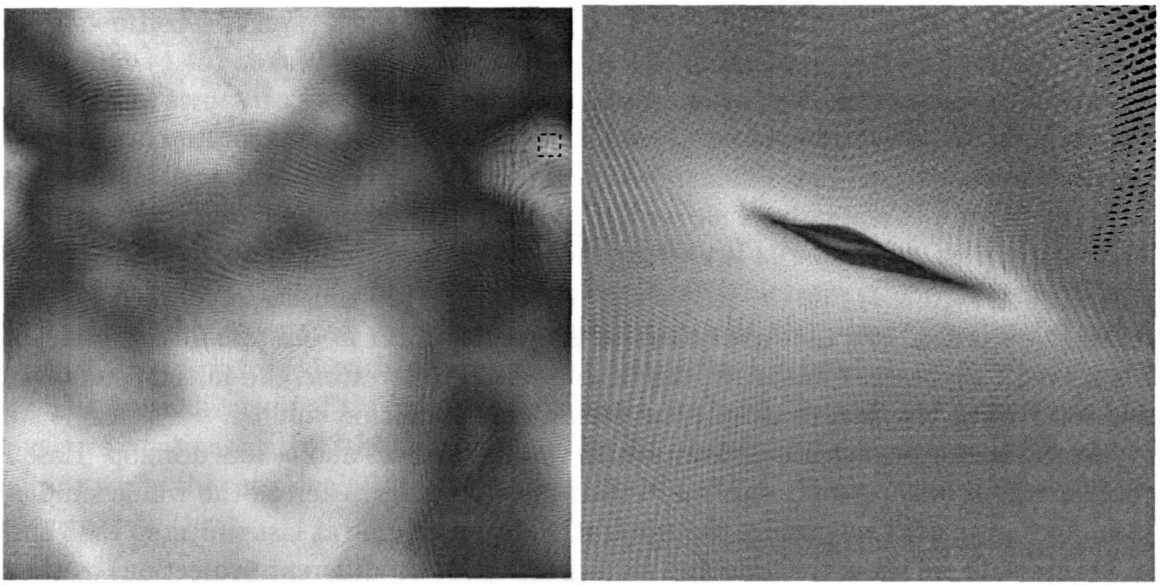

Figure 4-12: Left panel: initial conditions, at redshift $z = 350$. Density is projected into a plane from a slice of the whole simulation volume covering the whole width of the simulation box (22.56 comoving parsec) in the image dimensions but having thickness one quarter of the simulation volume. Right panel: the first caustics forming as a fold completing its first crossing of the center of the coordinates at redshift 130. The volume shown is a cube of the size of 30 code units. This perturbation is at the position shown by the small rectangle in the left.

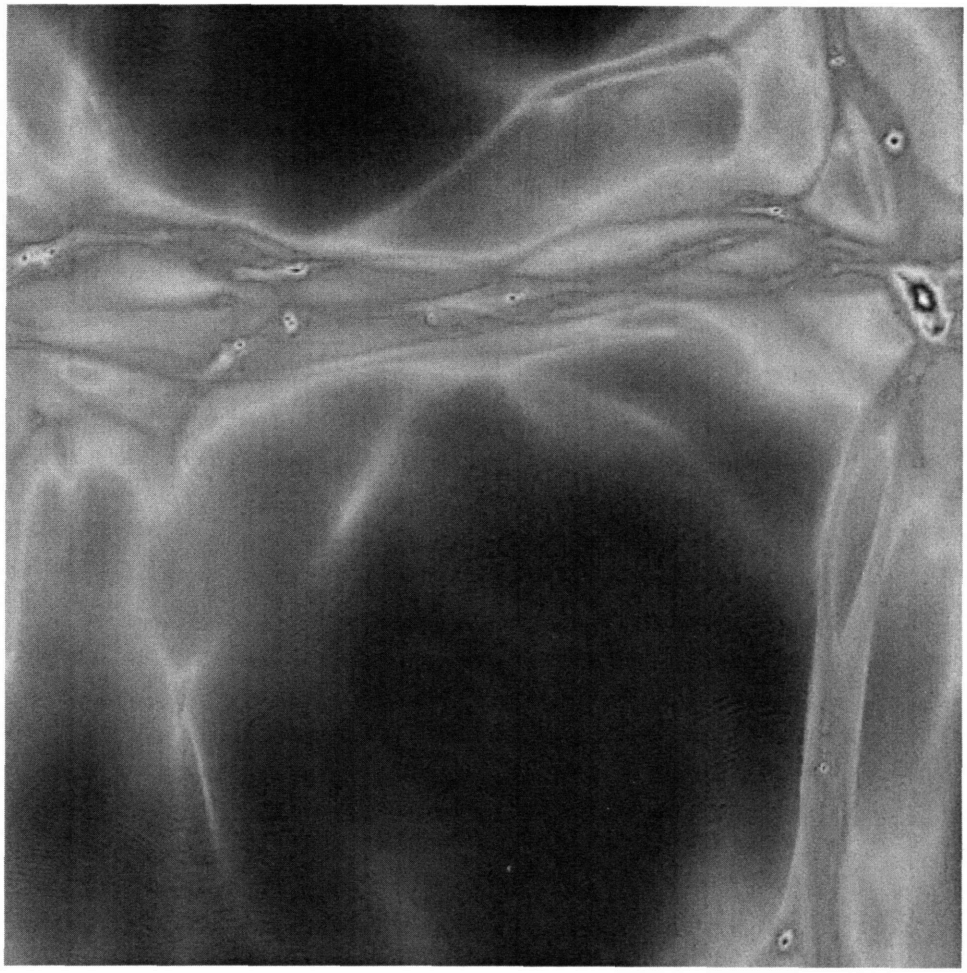

Figure 4-13: Matter density of whole simulation volume projected along y-direction at last timestep. Many halos have formed and are now undertaking frequent mergers. The massive halo at the right upper corner is halo I.

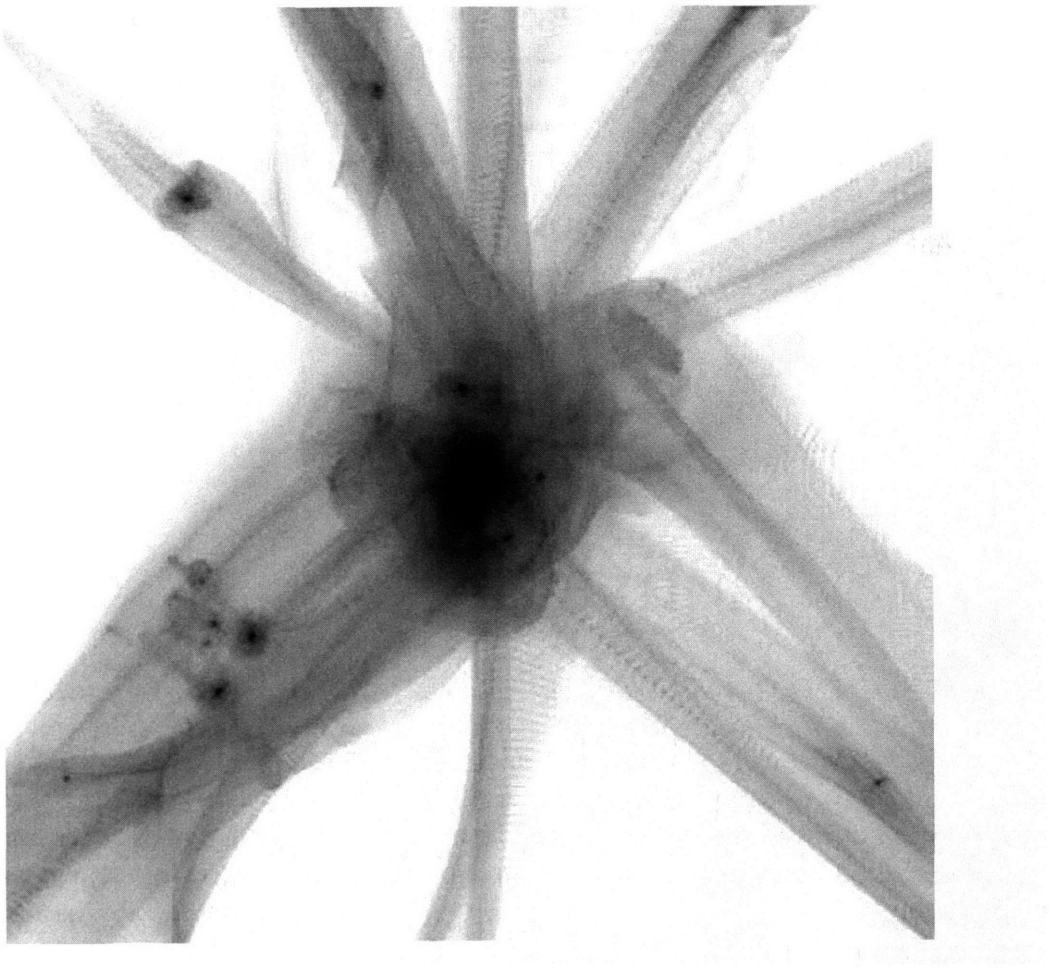

Figure 4-14: Halo I, whose formation is shown in Figure 4-12 is shown here in the cube of side 5.6×5.6 comoving parsec (200×200, in code units). This is the last timestep, redshift $z = 50.2$, projection along x-dimension.

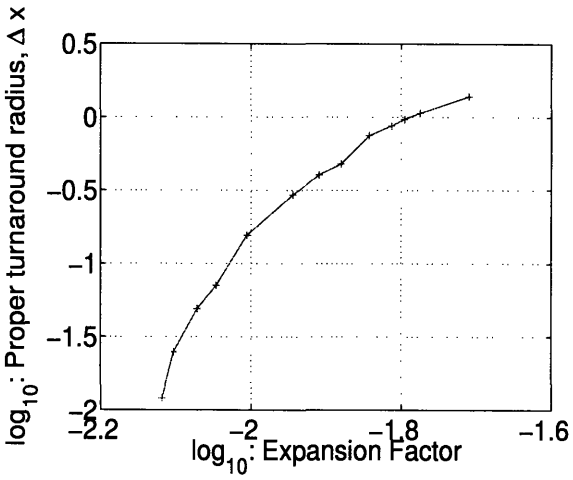

Figure 4-15: Dependence of the expansion factor on timestep in the P^{3}M caustic simulation.

One can roughly measure the first turnaround radius as a function of expansion factor as shown in Figure 4-15. This plot does not fit a power-law dependence. At small expansion factors the symmetry of the caustic is cylindrical, which by equation (4.22) yields a steeper dependence of turnaround radius as a function of expansion factor [the $n_{\rm sym} = 2$ case in equation (4.22)], than the case of spherical symmetry ($n_{\rm sym} = 3$). Equation (4.22) breaks down for $\epsilon_n = 0$. However, as expected the slope $d \ln r_{\rm ta}/d \ln a$ becomes more shallow as the matter distribution changes its symmetry from cylindrical to spherical under gravity.

In contrast to the outer structure, the inner structure of the halo, whose evolution starts from the formation of the first caustic just before timestep 130 (redshift 135), at timestep 300 (redshift 114) already does not have any visible caustics (see Figure 4-16). This result is completely different from the PM-simulation, whose inner caustic has survived a number of dynamical times, as shown in Figure 4-8.

Another way to observe the destruction of the inner caustic is to look at the phase space plot presented in Figure 4-17. The plot shows all the points within some radial separation from halo I at timestep 300 and 1983 (redshifts $z = 114$ and 50.2). The caustic structure can be observed similar to Figure 4-10 as a set of enclosed shells. Notice however that since the initial power spectrum had more fluctuations per given number of particles in the P^{3}M simulation than in the PM simulation, the phase portrait in Figure 4-17 undergoes more destructive effect from the angle averaging of the convoluted structure than in Figure 4-10 for the PM-simulation. Nevertheless, the caustic structure is still observed in the outer parts of the halo. No structure is observable in the inner regions of the evolved halo, where the caustics structure may be erased due to the projection from the six dimensional phase space into the two dimensions of the figure.

The destruction of the inner caustics occurs only in the simulation, not in the real universe. The effects destroying caustics were analyzed in Section 4.2.7. As we will conclude shortly, the inner caustic was destroyed by particle discreteness.

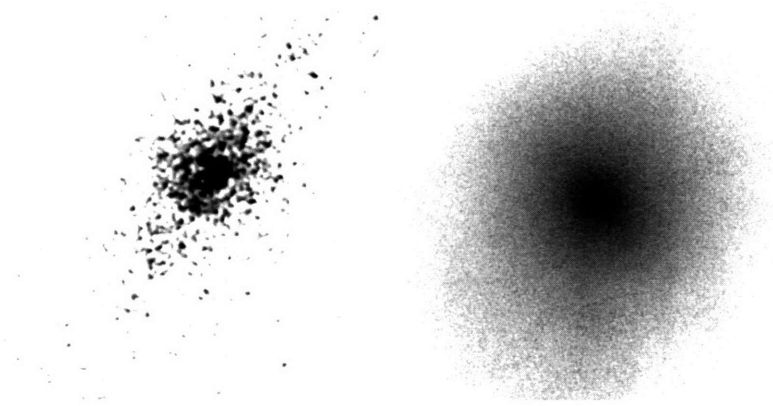

Figure 4-16: The inner structure of halo I. Left: Timestep 300 (redshift $z = 114$). The cubic volume shown is 1×1 in code units (0.028 pc on a side) and contains 21002 particles. The inner caustics are not visible either due to projection or folding from the phase space to the physical space, or they are already destroyed. Right: Timestep 1983, redshift $z = 50.2$, a cube of size 0.197 pc, containing 3.03×10^6 particles. Different scales are used for the brightness of the two images.

The destruction is probably caused by two-body relaxation of the simulation particles, equation (4.42). If the number of simulation particles per inner caustic is small enough the caustic is destroyed after a few dynamical times. By our timestep (4.54) criterion, one orbit is completed by the particles in the densest region in $N_{\rm orb}^{\rm min}$ timesteps, as given by equation (4.55). The caustics whose turnaround surface is sampled by N_c particles will therefore be destroyed in

$$\Delta N_{\rm step}^{\rm P3M} = \frac{t_{\rm relax}}{dt} = \frac{N_c}{\pi I} N_{\rm orb}^{\rm min} = \frac{N_c}{8 I \sqrt{\eta_t}} \ \text{timesteps}, \tag{4.58}$$

where we used equations (4.42, (4.53) and (4.54). Using $\eta_t = 0.05$ and $I \approx \ln(N_c)$, if the caustic contains 10 particles, we find that such a structure does not survive even a few timesteps.

In order to find whether two-body relaxation is responsible for the destruction of the caustics in the simulation, we need to find the number of particles in the latest surviving caustics. Let us immediately notice that whatever the reason is, it must be due to one of the differences with our PM simulation, where the inner caustics survived many dynamical times. Those differences are listed in Section 4.3.

Initially, as is visible from Figure 4-12 the inner caustic consists of at least a thousand particles. This number of particles in the innermost caustic reduces with time since the portion of the caustic surface closest to the center of the perturbation has the least dynamical time due to the higher density closer to the center. The inner portion of the innermost caustic surface therefore undertakes more revolutions around the center than the outer one. Thus, the number of particles *per revolution* of the caustic surface decreases with time.

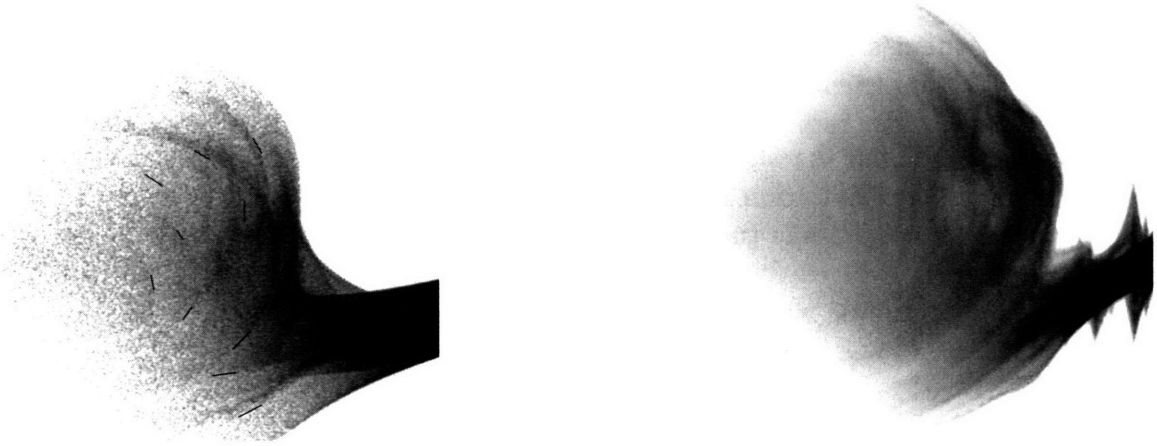

Figure 4-17: The phase space scatter plot in dimensions $(\ln r, v_r)$ now analogous to Figure 4-10 now shown for halo I of P^3M simulation. Left panel: timestep 300 ($z = 114$) a few dynamical times after the start of the inner caustic formation. The radial separation covers the range $r \in (0.1, 20)$ code units. Some structure still visible despite the projection effects is highlighted with the lines. Right panel: timestep 1983 ($z = 50.2$), radial separation range shown covers $r \in (1, 150)$ code units.

We can find the number of particles per caustic by plotting the z-Eulerian coordinate as a function of Lagrangian coordinates and seeing where the caustics are by examining $\partial z/\partial \mathbf{q}$. In Figure 4-18 we present the plot of Eulerian coordinate z (vertical axis) as a function of Lagrangian coordinates q_y and q_z in the plane in Lagrangian coordinates perpendicular to the x-direction and crossing a fixed particle in the core of halo I chosen at timestep 1383, $z = 63.6$, so that it has the minimum separation from the center of mass and minimum relative velocity with respect to the velocity of the center of mass. This figure is fully analogous to Figure 4-3 in Section 4.2.1, where a model Eulerian transformation was analyzed. However, in Figure 4-18 we take instead the real transformation given directly by the N-body simulation. The caustics are present in this plot as the onion rings enclosing one another in Lagrangian space. It is clearly visible in the plot that the caustic rings become less and less prominent closer to the central particle within the halo. The area of the projection of the surface onto the horizontal plane is the number of particles in the segment of the surface.

One can deduce the approximate number of particles in the shell caustics by analyzing the rings in the two dimensional cross section of the surface shown in Figure 4-18 by a plane along the q_y and q_z directions: as we know the distance in Lagrangian space directly transforms into the number of particles, since there is exactly one particle within each of the cells in Lagrangian space. In Figure 4-19 we present the cross sections along the y and z dimensions. Each of the points in both curves represents a particle. The number of particles in a caustic is deduced by multiplying the number of particles in a peak corresponding to the caustic $N_{\rm peak}$ by the surface area of the sphere of radius equal to Lagrangian separation $\Delta q = |\mathbf{q} - \mathbf{q_c}|$

from the center of the halo. Indexing the caustics by label i, we have,

$$N_c(i) = 4\pi[\Delta q(i)]^2 N_{\rm peak}(i) \ . \tag{4.59}$$

Using this formula and the observation from the figure that for the closest caustic to the center $i = 1$ the separation is $\Delta q = 14$ spacings of Lagrangian mesh, we find that the total number particles in that caustic is 246. This caustic is the innermost one to survive the highly evolved dynamical state. Combining equations (4.58) and (4.55) we find that this caustic will survive about

$$\frac{\Delta N_{\rm step}^{\rm P3M}}{N_{\rm orb}^{\rm min}} = \frac{N_c}{16\pi I} \approx 0.9 \tag{4.60}$$

more orbits, in other words this caustic will be destroyed in the next revolution. We deduce that the inner region of the halo in Figure 4-19 does not show caustics because they are destroyed due to the numerical finite resolution artifact of two-body relaxation.

It is interesting that the number of particles per density perturbation has decreased from $(40\pi)^3$ in the initial conditions to just 246 particles per smallest structure. The huge decrease in the the number of particles per structure was not observed in our PM-simulation described in Section 4.3.1, where the innermost caustic remains stable over many dynamical times and its oscillation amplitude in physical space does not significantly change. As we listed in Section 4.3.2, there are three differences of the PM run with the P^{3}M simulation runs. It is not clear at this point which of these three differences led to such a significant change in evolution of the number of particles per caustic. Further tests are necessary in order to establish the reason.

By analyzing the trajectories of the particles adjacent to the selected particle $\mathbf{q_c}$ in the core of halo I in Lagrangian space, we can check the prediction of the self-similar model that with our initial power spectrum, the apoapsis of the trajectory behaves according to equation (4.25), which for the power spectrum slope $n = -3$ gives $q = -\infty$. Equation (4.25) was found using adiabatic approximation valid as averaged over many orbital rotations. This implies by equation (4.25), that if the result is true the apoapsis separation should asymptotically approach zero.

Figure 4-20 shows the relative trajectories of selected particles. The z-component of Eulerian separation of the selected particles from the fixed particle $\mathbf{q_c}$ in the core of halo I is plotted as a function of timestep. We chose particles near $\mathbf{q_c}$ in Lagrangian space. We see that formation of halo I gives rise to the formation of a plateau in the left plot, surrounded by walls of the oscillations at the boundaries. As we see, with the growth of the halo, the plateau widens, however the apoapsis distance does not reduce, as seen in the right panel of Figure 4-20. This result seems to contradict the conclusion of equation (4.25) that the apoapsis distance should asymptotically approach zero.

The apoapsis distance defines the size of the innermost caustic. When in the P^{3}M simulation the apoapsis distance goes to zero, the number of particles in one revolution of the caustic surface reduces indefinitely until the caustic is destroyed. In

the PM-simulation, where $\epsilon_n = 1$, and $q = 0$, the apoapsis distance does not shrink so rapidly instead reaching a constant value of a few units of grid spacing (much larger than that in Figure 4-20 for P^3M). If the apoapsis distance determines the size of the innermost caustic, the latter will not continue to shrink, and will therefore survive for many dynamical times, as was indeed observed in Section 4.3.1.

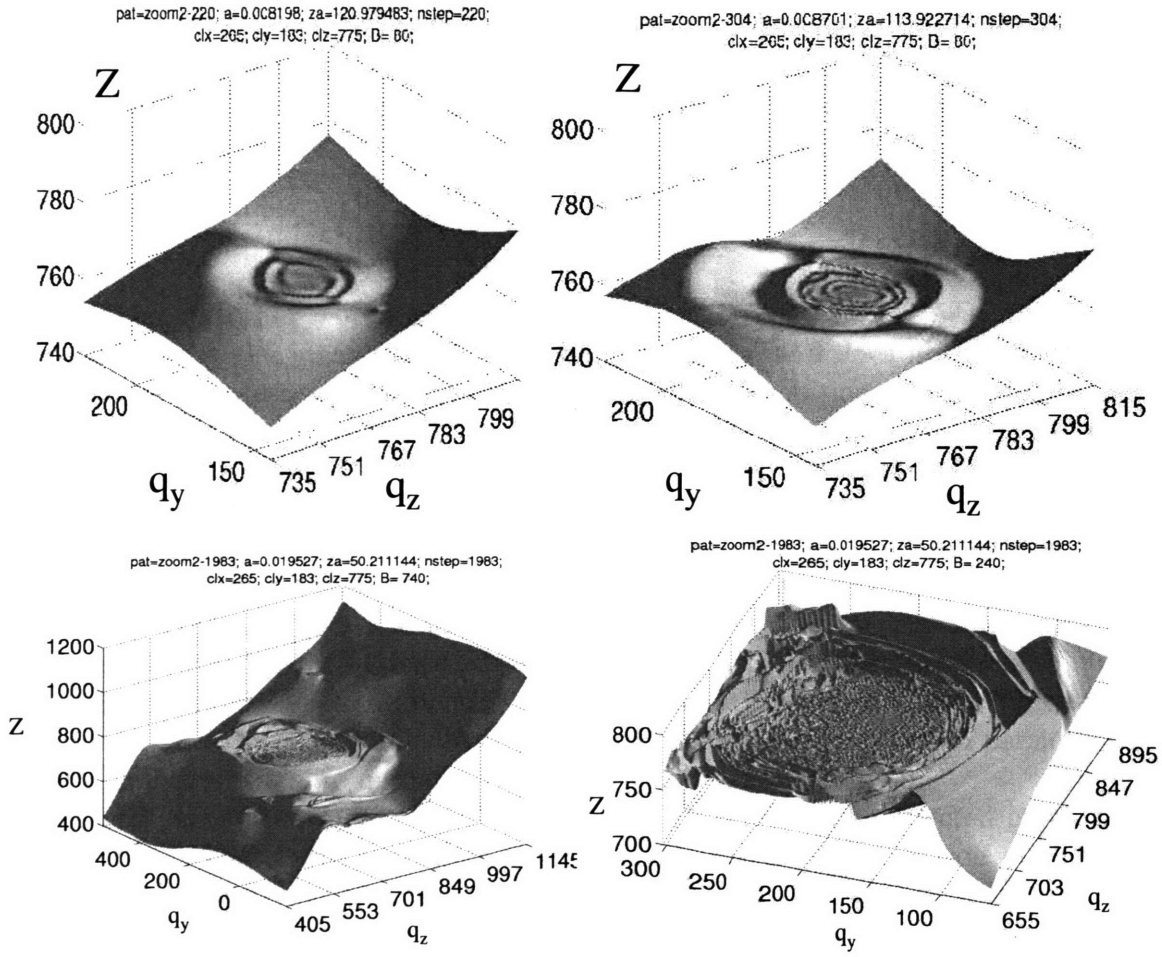

Figure 4-18: Eulerian z-coordinate of the particles within the (q_y, q_z)-plane crossing the central halo particle in the Lagrangian space as a function of the position in Lagrangian plane. Shown in each panel is the rectangular domain in Lagrangian (q_y, q_q) centered on a selected central halo particle. The dimensions of the horizontal axes are in grid spacings, and those in the vertical Eulerian Z-axis are in code units $\Delta x = 0.028$ pc. Points on this surface where $\partial Z/\partial q_z = 0$ belong to the caustic surface.
Upper Left: Timestep 220 ($z = 121$), Upper Right: Timestep 314 ($z = 113$)
Bottom Left: Timestep 1983 ($z = 50.2$), Bottom Right: Same timestep but zoomed in.

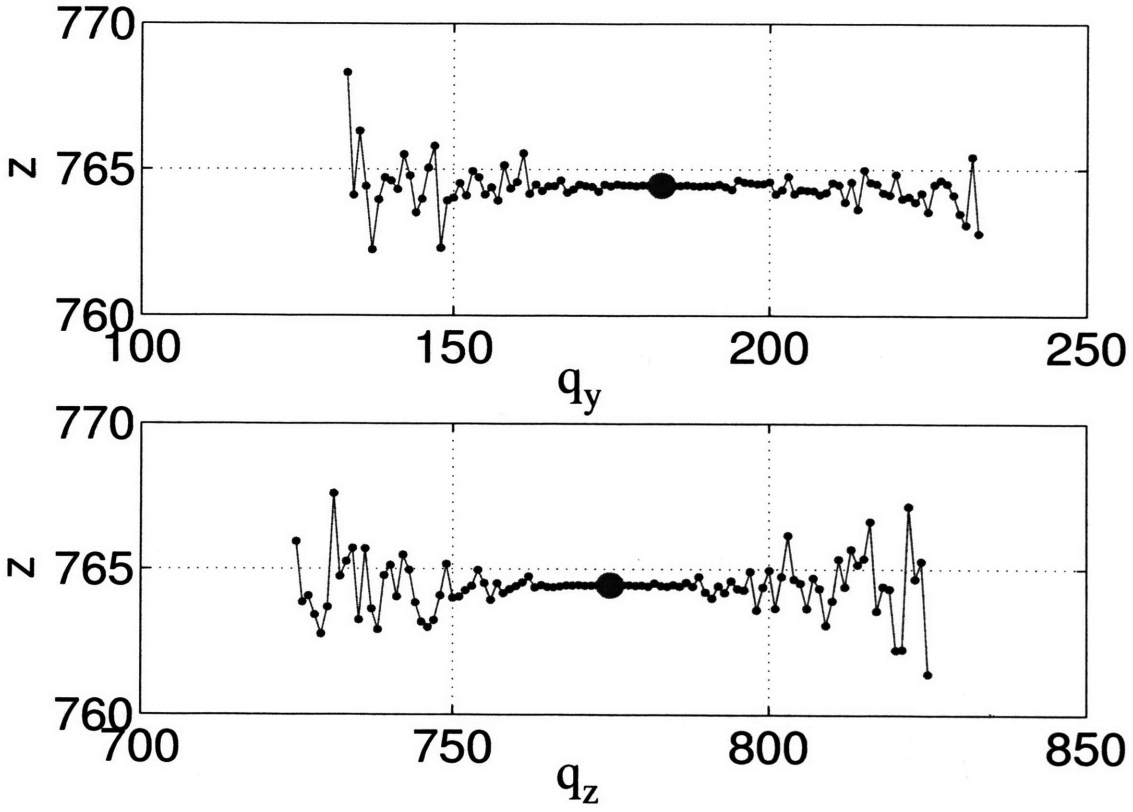

Figure 4-19: Timestep 1983 ($z = 50.2$), halo I. The cross section of the surface shown in Figure 4-18 by a plane along y and z-directions. Oscillations occur at every caustic.

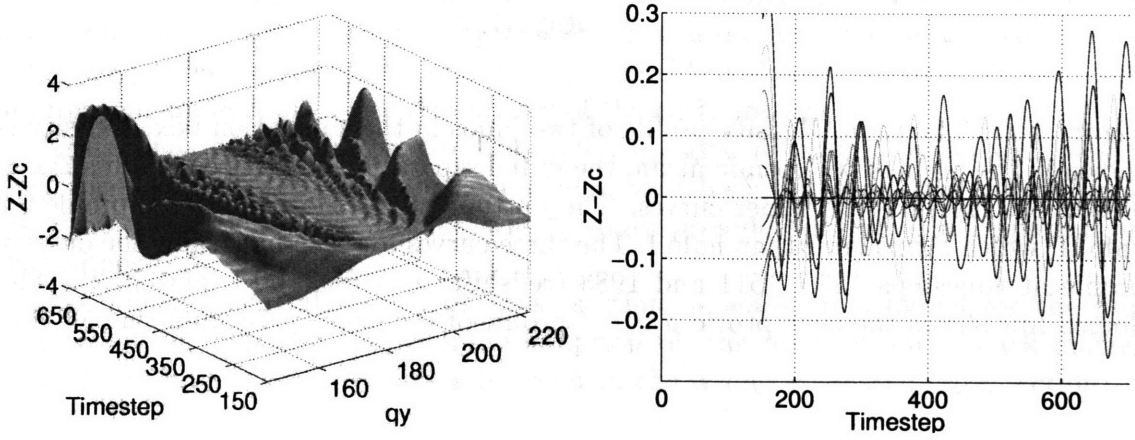

Figure 4-20: Left: Z trajectories of a set of adjacent particles labeled by q_y, while $q_x = q_{cx}$ and $q_z = q_{cz}$. Note that this plot is different from the plots in Figure 4-19 which do not present any time dependence of the structure. Right: The same quantity is plotted as a function of timestep for a few particles covering a uniformly sampled range along Lagrangian y-direction. The range of the timesteps presented starts with the formation of halo I at the redshift 130.

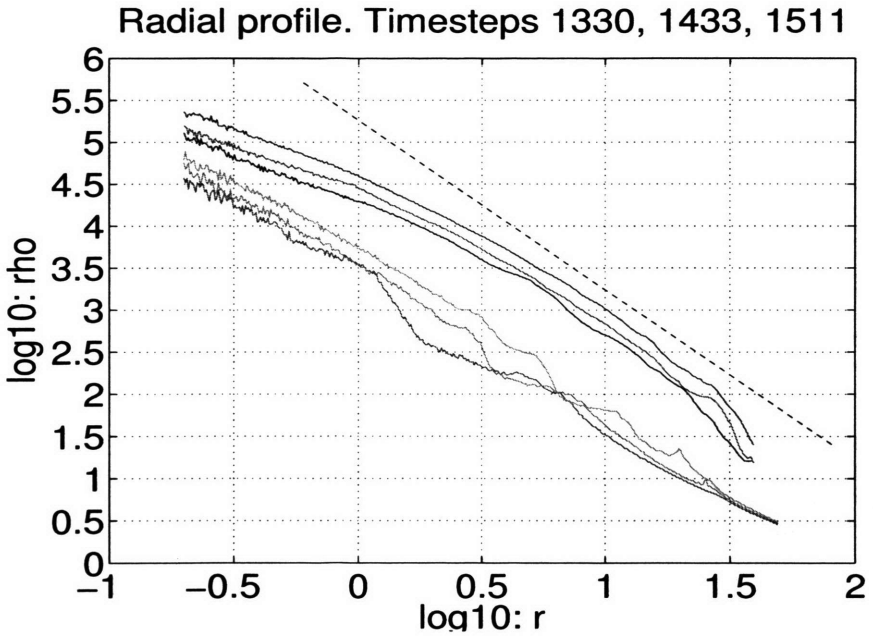

Figure 4-21: The radial density profile of two halos in the simulation box. Density is shown in the units of the cosmic mean, the radial separation is in the code units $\Delta x = 0.0282$ pc. Halo I is the upper curves. The bottom curves show the radial profile for halo II, which forms later than halo I. The three curves for each halo show the density profile at timesteps 1330, 1511 and 1983 (redshifts $z = 65.2$, 60.1, and 50.2). The dotted line shows the asymptotic $\rho \propto r^{-2}$ profile of the exact self-similar model [25].

The self-similar model predicts that the slope of the halo radial density profile evolves to a certain number based on the power-law slope in the initial conditions [see equations (4.21) and (4.23)]. The P^3M simulation, for which $\epsilon_n = 0$ should then have $\rho \propto r^{-2}$. In Figure 4-21 we present the radial density profiles for two of the halos at the same set of redshifts.

Halo I shows agreement with the model power spectrum $\rho \propto r^{-2}$ for $5 \leq r/\Delta x \leq 25$. This halo was the first to form in the simulation volume and evolved more than 20 crossing times. The radial profile changes at the outer caustic at $r \approx 30\Delta x$.

Halo II went into the nonlinear regime much later than than halo I, therefore at each redshift it is younger than halo I. We observe that for the younger halo, the density profile does not follow the predicted slope, instead following another power-law $\rho \propto r^{-1.6}$. However this can probably be explained by the fact that this halo is not sufficiently evolved by gravitation. Indeed from the phase space portrait of this halo we know that by the last timestep it has undertaken only about two or three shell crossings, in addition this halo does not have apparent spherical symmetry at this stage. Instead it has a more apparent cylindrical symmetry, for which one would expect $\rho \propto r^{-1}$.

4.3.4 Implications for Direct Detection CDM Search Experiments

In the ongoing efforts to detect the dark matter experimentally, many collaborations, such as the Cryogenic Dark Matter Search (CDMS) [1] and the DArk MAtter experiment (DAMA) [4], have produce limits on the neutralino cross section as a function of mass. These quantities are fundamental for our understanding of neutralino dark matter and for particle physics models. If the neutralino mass and cross section of weak interaction with nucleons (m_χ, σ_χ) are measured correctly in an experiment, they will produce a constraint on supersymmetric models of particle physics, and improve our understanding of the physics of the early universe.

Neutralinos can be detected by the kick they give to atomic nuclei with which they elastically scatter. Once nuclear recoils have been detected, one can measure the time correlation of the events, which would provide additional information on neutralinos.

The inferred neutralino-nucleon cross section depends on a model for the flux of dark matter particles. A wrong assumption on the distribution of dark matter particles in phase space will generally result in the wrong estimated cross section.

Locally Uniform Matter Distribution

The mean neutralino scattering rate per target detector nucleon is obtained from the phase space distribution function of neutralinos in the local rest frame of the detector $f(\mathbf{r}, \mathbf{v}, t)$ in the position and velocity space $(\mathbf{r}, \mathbf{v})$ as a function of time t by

integration over velocities

$$\Gamma = \int f(\mathbf{r}, \mathbf{v}, t)\, \sigma_\chi(v)\, v\, d^3\mathbf{v}, \quad \text{sec}^{-1} \tag{4.61}$$

where the cross section σ_χ of weak interaction between neutralino and nuclei is in general a function of the energy or the absolute relative velocity v of the colliding neutralino with respect to the target atom.

Although the dark matter density should depend on position and time, most analyses ignore these variations. In addition, the neutralino velocity is widely assumed to be Maxwellian distributed with corrections due to a finite escape velocity $v_{\text{esc}} \approx 600$ km s^{-1} of particles from the galatic halo and a correction due to the Earth's motion with velocity v_E relative to the galactic center. In other words, the assumption for the phase space distribution function in the Earth rest frame used to produce neutralino particle data limits (see [35] and [1] for more details) is

$$f(\mathbf{r}, \mathbf{v}, t) = n_0 f_{\text{b}v}, \quad \text{(wrong)} \tag{4.62}$$

where

$$f_{\text{b}v} = \frac{1}{(\pi v_0^2)^{3/2}}\, e^{-|\mathbf{v}+\mathbf{v_E}|^2/v_0^2}, \ |\mathbf{v}| < v_{\text{esc}}. \ \text{(wrong)} \tag{4.63}$$

Here $v_0 \approx 220$ km s^{-1} is a constant and n_0 is the locally uniform neutralino number density. The escape velocity v_{esc} is used to correct the phase space density for the particles that leave the galaxy due to their high speed; the phase space density vanishes for $|\mathbf{v}| \geq v_{\text{esc}}$.

Using equations (4.62) and (4.61), one can get the scattering rate per target nucleus. However, equation (4.61) can be simplified. The velocities of the local dark matter particles with respect to the detector are of the order of the Galaxy escape velocity v_{esc}. At such low velocities the cross section of weak interaction of neutralino with nuclei can be assumed to be constant and be brought outside the integral in equation (4.61). Similarly, one can factor out the average number density. The resulting expression for the detection rate in the local fluid element per target nucleon is

$$\Gamma = n_0\, \sigma_\chi\, \langle v \rangle\ , \tag{4.64}$$

where $\langle v \rangle \equiv \int f_{\text{b}v} v d^3\mathbf{v} / \int f_{\text{b}v} d^3\mathbf{v}$. The above expression was in fact used by [1] in order to get their limits on the neutralino particles. This expression gives the same detection rate anywhere within the solar neighborhood.

The CDMS Dark Matter Detection Experiment

We now analyze the (CDMS) experiment [1] where the assumption of a uniform matter distribution was used to derive upper limits on neutralino-nucleon weak interaction cross section.

The detector consists of germanium and silicon probes of masses respectively $M_0 = 1000$ g, and $M_1 = 200$ g, working simultaneously. The masses of silicon and germa-

nium probes yield the total mass of the detector $M = \sum_p M_p = M_0 + M_1$, where p is an index listing all the probes of the detector. The cryogenic temperature of the probes allow one to neglect thermal velocities of the detector nuclei while using equation (4.63). The Maxwell-Boltzmann distribution parameter values used are $v_E = 232$ km s^{-1}, $v_0 = 220$ km s^{-1}, the escape velocity $v_{\rm esc} = 600$ km s^{-1}; the neutralino local density is $\rho_\chi = 0.3$ GeV cm$^{-3}\,c^{-2}$. In the experiment described in [1] the exposure of 28.3 kg-day was acheived, in which no neutralino detection was observed.

The atomic number A or the number of nucleons in the detector atom provides the scaling for the spin-coherent scattering of the neutrinos by the atomic nuclei. The cross section of such scattering per nucleus scales as A^3, where the power of three is obtained by combining the factor of A^2 due to the low energy spin-coherent scattering while another power of A converts the cross section per nucleus to cross section per nucleon. The total cross section due to all the nuclei in the experiment equals

$$\mathcal{A}\sigma_0 \equiv \sum_p \left(\frac{M_p}{m_{\rm AU} A_p} A_p^3 \right) \sigma_0 \,, \tag{4.65}$$

where $m_{\rm AU}$ is the atomic unit mass and A_p are the atomic numbers of the isotopes of nuclei ^{28}Si and ^{73}Ge used in CDMS, where σ_0 is the cross section of neutralino interaction for one nucleon, being approximately the same for both silicon and germanium atoms.

The detection rate per unit mass of the experiment is

$$R_0 = \frac{n_0\, \mathcal{A}\, \sigma_0 \langle v \rangle}{M} \,. \tag{4.66}$$

Assuming $v_{\rm esc} = \infty$ and $v_E = 0$ we have $\langle v \rangle = 2v_0/\sqrt{\pi}$. In order to find the detection rate for the finite v_E and $v_{\rm esc}$, one only needs to make a correction to average neutralino velocity $\langle v \rangle$ due to the difference in phase space distribution function from Maxwellian in Earth's rest frame. This correction, which will be denoted in our notations by $\gamma(v_E, v_{\rm esc})$ equals the right-hand side of equation (3.5) of [35] and depends exclusively on $v_{\rm esc}$ and v_E. Let us list for reference

$$\begin{aligned} \gamma(0, \infty) &\equiv 1 \\ \gamma(0, v_{\rm esc}) &= 0.99695 \\ \gamma(v_E, v_{\rm esc}) &= 1.3325 \,, \end{aligned} \tag{4.67}$$

where the values of $v_E, v_{\rm esc}$ are listed above, as used for the experiment. The average neutralino velocity, corrected for Earth's motion and finite escape velocity, is

$$\langle v \rangle_0 = \frac{2\gamma(v_E, v_{\rm esc}) v_0}{\sqrt{\pi}} \tag{4.68}$$

For example, the average neutralino velocity $\langle v \rangle_0$ in the Earth's rest frame and in the

galaxy rest frame $\langle v \rangle_G$ equals

$$\begin{aligned} \langle v \rangle_0 &= (2/\sqrt{\pi})\, v_0 \gamma(v_E, v_{esc}) \approx 330.78 \text{ km s}^{-1} \\ \langle v \rangle_G &= (2/\sqrt{\pi})\, v_0 \gamma(0, v_{esc}) \approx 1.1249 v_0 \approx 247.48 \text{ km s}^{-1} . \end{aligned} \tag{4.69}$$

The detection of neutralinos in the experiment is a Poisson process with expected locally uniform Poisson average of the total number of detections

$$\lambda_0 = \mathcal{E} R_0 = F_0 \frac{\mathcal{E}\, \mathcal{A}}{M} \frac{\sigma_0}{m_\chi}, \text{ where } F_0 \equiv \rho_0 \langle v \rangle_0 , \tag{4.70}$$

the neutralino background density $\rho_0 \equiv n_0 m_\chi$, and $\mathcal{E}$ is the exposure usually given in units of kg-day.

The probability of detecting N events from a Poisson probability distribution with mean λ_0 is given by

$$P_{\lambda_0}(N) = \frac{\lambda_0^N}{N!} \exp(-\lambda_0) . \tag{4.71}$$

In particular, the probability of detecting zero events is $P_{\lambda_0}(0) = \exp(-\lambda_0)$. The absence of detections $N = 0$ in an experiment provides the upper bound λ_0^+ on the uniform Maxwellian Poisson average λ_0

$$\lambda_0 < \lambda_0^+, \quad \lambda_0^+ \equiv \ln[1/(1 - \text{C.L.})] = 2.303, \text{ for C.L.} = 90\% . \tag{4.72}$$

directly following from inequality $P_{\lambda_0}(0) \geq 1 - \text{C.L.}$. Equation (4.72) provides the upper limit on the ratio (σ_0/m_χ) in equation (4.70). Using the numbers we presented above we find the upper limit on cross section for the CDMS experiment based on the assumption of uniformity of spatial matter distribution

$$\sigma_0 \leq \sigma_0^+ = 5.03 \times 10^{-42} \text{ cm}^2 \times \left(\frac{m_\chi}{100 \text{ GeV}\, c^{-2}} \right) \tag{4.73}$$

which agrees with the number presented in Figure 4 of [1] for $m_\chi = 100$ GeV c^{-2}.

In addition to the above, a new CDMS II experiment [2] has produced a lower value of $\sigma_0 \leq 4 \times 10^{-43}$ cm^2 for $m_\chi = 60$ GeV with C.L. = 90% based on 19.4 kg-d of exposure. We are uncertain how such a low exposure leads to a low upper limit on the cross section and assume that this is due to combining multiple datasets.

Effect of Small-Scale Fluctuations

In the real dark matter, the caustics introduce a strong dependence of the number density on position, as described previously in this Chapter. In general, a phase space distribution function of a CCF may be separated into the spatial and velocity parts

$$f(\mathbf{r}, \mathbf{v}, t) = n(\mathbf{r}, t)\, f_v(\mathbf{r}, \mathbf{v}, t) , \tag{4.74}$$

where $\int f_v(\mathbf{r},\mathbf{v},t)\,d^3\mathbf{v} = 1$, resulting in

$$\Gamma(\mathbf{r},t) = n(\mathbf{r},t)\,\sigma_\chi\,\langle v(\mathbf{r},t)\rangle\,. \tag{4.75}$$

instead of equation (4.64). There is a large difference between these two equations, since the former completely ignores the local density and velocity perturbations on the scales of the solar system.

Doing similar manipulations as we did above starting from the general CCF phase space distribution function given by equation (4.74) and the rate given by (4.75) we find that for a given experiment positioned at $(\mathbf{r},t)$, the Poisson average of the number of detections is given by

$$\lambda(\mathbf{r},t) = \frac{\mathcal{E}\mathcal{A}\sigma_0}{M}\int f(\mathbf{r},\mathbf{v},t)\,v\,d^3\mathbf{v}\,. \tag{4.76}$$

Consider a cubic volume in the universe with proper side length L. The phase space distribution function for the neutralino particles in this volume is

$$f(\mathbf{r},\mathbf{v},t) = \sum_{j=0}^{N_\chi-1}\delta^3(\mathbf{r}-\mathbf{r}_j)\,\delta^3(\mathbf{v}-\mathbf{v}_j)\,, \tag{4.77}$$

where N_χ is the total number of neutralinos inside the volume. If the box size is large enough so that the perturbations on the scale of the box are still linear N_χ is well approximated by $N_\chi = 3H_0^2\Omega_m L^3/8\pi G m_\chi$. Suppose now that the volume is subdivided into $N_c = n_c^3$ cells of spacing $\Delta x_c = L/n_c$. Each cell contains ${N_\chi}^c$ particles so that $N_\chi = \sum {N_\chi}^c$, where $c \in [0, N_c)$ is the index listing each cell. The average number of neutralinos detected in a cell c is then

$$\lambda_{\rm cell}(c,t) \equiv \frac{1}{(\Delta x_c)^3}\int_{\rm cell} d^3\mathbf{r}\;\lambda(\mathbf{r},t)\,. \tag{4.78}$$

As we know from Section 4.2.1, the dynamics of the neutralino dark matter is described as the dynamics of a CCF. The neutralino velocities can therefore be locally expanded in a Taylor series at any position [c.f. equation (4.9)]. If the dark matter CCF fluid elements are sufficiently well sampled by the simulation particles as is the case in the caustic simulations described earlier in this Chapter, then the structure and topology of the real dark matter CCF is preserved when simulation particles are introduced instead of the dark matter particles. The velocity $v_{\rm sp}$ of a simulation particle at a given position is defined as the velocity of the center of mass of the neutralinos contained in the local fluid element of the CCF. Plugging equations (4.76) and (4.77) into (4.78), we have

$$\lambda_{\rm cell}(c) = F_{\rm sym}(c)\frac{\mathcal{E}\mathcal{A}}{M}\frac{\sigma_0}{m_\chi} = \frac{F_{\rm sym}(c)}{F_0}\lambda_0, \tag{4.79}$$

where

$$F_{\rm sym}(c) \equiv \frac{m_{\rm sp}\sum_{j=0}^{N_{\rm sp}(c)-1} v_{{\rm sp}\,j}}{(\Delta x_c)^3} = \frac{m_{\rm sp}N_{\rm sp}(c)\langle v_{\rm sp}\rangle_c}{(\Delta x_c)^3}\,, \tag{4.80}$$

and $N_{\rm sp}(c)$ is the number of simulation particles of mass $m_{\rm sp}$ in c. It is much smaller than the number of neutralinos $N_\chi(c)$ in the same cell. Equation (4.79) for the Poisson average of the detections is valid when the smallest caustics are resolved by the simulation particles and the size of the detector is comparable to the cell spacing.

A comparison of equations (4.79) and (4.70) demonstrates the difference in using a spatially a uniform Maxwellian distribution instead of the real non-uniform matter distribution for the expected Poisson average of the detections in the experiment. The ratio of the mass flux measured by a detector placed within a cell in the simulation volume to the flux computed on the assumption of uniform Maxwellian distribution is

$$\frac{F_{\rm sym}(c)}{F_0} = \left[\frac{L^3/N_{\rm sp}}{(\Delta x_c)^3}\right]\frac{N_{\rm sp}(c)\langle v_{\rm sp}\rangle_c}{\langle v\rangle_0}\,, \tag{4.81}$$

where the quantity in square brackets is the ratio of the average volume per particle divided by the volume of a cell.

If we knew the position of the detector inside the simulation volume, we would find the upper limit on the cross section by using equations (4.70) and (4.71), where instead of the spatially uniform F_0 we would use $F_{\rm sym}(c)$ which varies from cell to cell, depending on the local mass flux distribution. Since there has been no detection so far, the local neutralino mass flux is unknown and the local Poisson average detection rate $\lambda_{\rm cell}(c)$ is unknown, instead being a random variable. The probability to make N detections in an experiment whose position inside the box is unknown is given by

$$P(N) = \frac{1}{N_{\rm c}}\sum_{c=0}^{N_{\rm c}-1} P_{\lambda_{\rm cell}(c)}(N) = \frac{1}{N_{\rm c}}\sum_{c=0}^{N_{\rm c}-1}\frac{(\lambda_{\rm cell}(c))^N}{N!}e^{-\lambda_{\rm cell}(c)}\,. \tag{4.82}$$

In analogy with equation (4.72), the absence of detections in the CDMS experiment implies the upper limit λ_s^+ on λ_0 of equation (4.70) set by condition $P(0) \geq 1 - \text{C.L.}$ yielding

$$\lambda_0 \leq \lambda_s^+\,, \tag{4.83}$$

where λ_s^+ is defined by

$$\frac{1}{N_{\rm c}}\sum_{c=0}^{N_{\rm c}-1}\exp\left[-\frac{F_{\rm sym}(c)}{F_0}\lambda_s^+\right] \equiv 1 - \text{C.L.}\,. \tag{4.84}$$

For a uniform matter distribution $\lambda_{\rm cell}(c) = \lambda_0$ and $\lambda_s^+ = \lambda_0^+$. Given the fluxes $F_{\rm sym}(c)$ in the above equation, the above equation is easily solved numerically since the function in the left-hand side is monotonic with λ_s^+.

Upper Limit on Neutralino Cross Section

The simulation of matter on small scales described in Section 4.3.3 was evolved to timestep 1983 (redshift $z = 50.2$), at which point it was stopped because the matter distribution on the scales of the box size became nonlinear due to the finite size of the simulation box. In order to evolve the matter distribution to the present time by an N-body simulation one needs to have a sufficiently large simulation box so that it includes the $z = 0$ nonlinearity length scale equal to 5 h^{-1} Mpc [15]. Such a large volume is impossible to simulate with the same mass resolution as our P^3M simulation, since it would require at least 1.7×10^{25} particles.

In order to apply the simulation results for the dark matter detection experiments that are being carried out at present, we need to extend the output of the simulation from $z_* \geq 50.2$ to $z = 0$. Since it is not possible to evolve the N-body simulation beyond $z_* = 50.2$ we apply a *frozen clustering* model to analytically evolve the matter and velocity distribution to $z = 0$. Suppose we are given particle positions and velocities from an output of the simulation at some redshift z_*. Under this model the spatial clustering remains the same in comoving coordinates while the velocities are rescaled by a constant factor to match their present time values. This model is motivated by the fact that in the strongly nonlinear regime a scale-free hierarchy is expected to reach an asymptotic quasi-stable state of self-similar evolution.

Under the frozen clustering assumption we rescale the velocities of simulation particles from the output of the simulation at redshift z_* so that the resulting average velocity in the simulation volume rest frame matches the assumption $\langle v \rangle = \langle v \rangle_G$ of [1]. We then correct the velocity distribution for the Earth's velocity v_E with respect to the Galaxy rest frame by adding an arbitrarily chosen vector of this magnitude to be the direction of Earth's motion and subtracting this vector from the velocity vectors of each particle. Setting the expansion factor to unity at this point, we have a realization of the dark matter distribution at the present time. For example for the output at $z_* = 50.2$ the average magnitude of particle velocities in the rest frame of the simulation volume is $\langle v \rangle = 20.8$ km s^{-1}. The velocities are rescaled by a factor of ≈ 11.9 in order to exactly match $\langle v \rangle_G$ given by equation (4.69). After the correction due to Earth's motion, the average magnitude of the neutralino velocities is 345.4 km s^{-1} close to the value $\langle v \rangle_0$ given by equation (4.69).

By evolving the matter distribution using the frozen clustering model we introduce an error in the present-day matter distribution. We can test the effect of this error by comparing the results obtained using different z_* which is possible since the particle data were saved many times during the run. In the discussion below we will use the outputs at timesteps

Timestep	1	40	80	150	600	900	1382	1983
Redshift z_*	330	234	175	131	95	80.5	63.6	50.2

(4.85)

from which we evolve particle distribution to $z = 0$ using the frozen clustering model.

In Figure 4-22 we present the mass flux probability distribution functions for each z_*. Equation (4.81) is used to determine the mass flux for each HC cell. Then

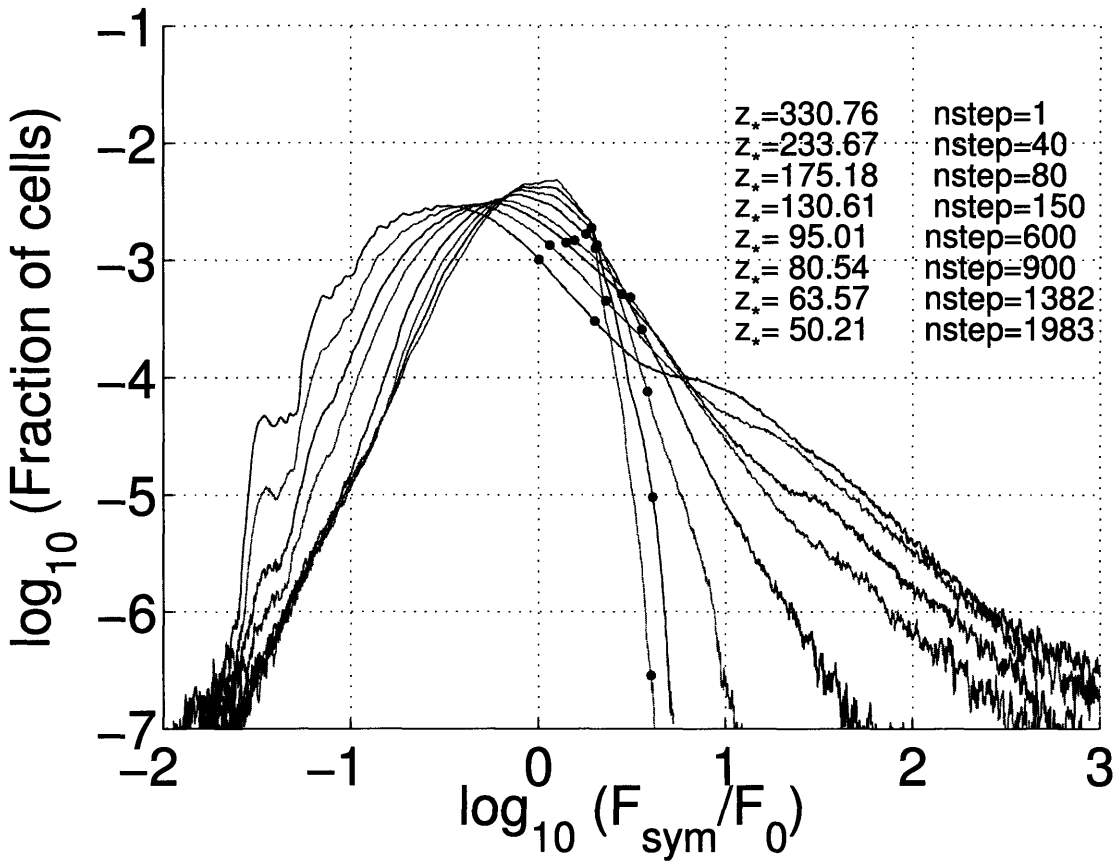

Figure 4-22: Binned mass flux probability distribution function in the simulation box as a function of redshift z_* beyond which the particles are assumed not to move. The innermost curve corresponds to the highest z_*. The pairs of points shown on the curves were used to measure the slope further in the text.

the histogram of mass flux $F_{\rm sym}$ with equally spaced bins in logarithmic space is constructed. In contrast with the constant flux for each cell, as expected in [1], we observe a range of possible flux values depending on the position within the volume.

The $z_* = 330$ curve corresponds to the initial conditions given by Gaussian fluctuations in density and velocity. The behavior of the mass flux ρv in the initial conditions is dominated by the perturbations in the initial velocity field. The mass flux therefore is a random variable whose probability distribution function $dN(F)$ is

$$\frac{dN(F)}{d\ln F} \propto F^3 \exp(-CF^2) \tag{4.86}$$

where C is a constant. This relation is indeed followed in the figure. It is a bad approximation to use such a high starting redshift for the frozen clustering model evolution, since the matter perturbations at this redshift were linear and therefore did not enter the asymptotic stage of the self-similar evolution for which the frozen

clustering model is better applicable.

When matter becomes strongly clustered at the later timesteps, the range of the mass fluxes broadens by orders of magnitude and the dependency at $F \gg F_0$ changes from an exponential cutoff to a double power law. The change in the behavior at high mass flux is caused by the formation of small scale structure such as caustics. The structure of caustics is poorly sampled by the HC cells whose spacing is ≈ 2.78 PM-density mesh cells. The power law changes its slope at high fluxes because these high flux cells correspond to the densest regions whose caustic structure is most unresolved by the HC-mesh cells. If the structure is resolved, we would probably have a single power law, which is increasingly the case at lower redshifts when most of the matter has not collapsed to small scales. One can see that the double power law effect is almost negligible for the earlier starting redshifts z_* for which the small scale structure has not yet collapsed to length scales that are smaller than the HC-cell spacing. The effect of the double power law likely therefore will be avoided when finer cells are used for the overdense cells. We predict that there is a single power law dependence valid when the appropriate mesh cells are used for sampling perturbations.

$$\frac{d\ln N(F)}{d\ln F} \propto F^{-A_{\rm F}}, \quad F \gg F_0\,, \tag{4.87}$$

where $A_{\rm F}$ is a constant. We measured the slope within the range $\ln F_{\rm sym}/F_{\rm sym} \approx 0.3$, where density perturbations and caustics are resolved with the HC cells. Using the selected points in Figure 4-22 we found that the value

$$A_{\rm F} \approx -2 \tag{4.88}$$

is approached from above as a_* increases as shown in Figure 4-23. It is not clear however whether this is an asymptotic value at high a_*.

In order to find the correction to the results [1] on the upper limit of the neutralino cross section due to spatial clustering, in Figure 4-24 we present the dependence of λ_s^+ on a_*. We observe that in the limit of high redshift the curve reaches the value $\lambda_0^+ \approx \ln[1/(1-\text{C.L.})] \approx 2.303$ as expected from a uniform Maxwellian distribution. The clustering evolves under gravity, causing λ_s^+ to deviate from its initial value λ_0^+ valid under the assumption of spatial uniformity growing to the higher values. Using the highest available $z_* = 50.2$, we find the upper constraint on λ_0

$$\lambda_0 \leq \lambda_s^+(50.2) \approx 4.1\,\lambda_0^+ = 4.1\,\ln[1/(1-\text{C.L.})]\,. \tag{4.89}$$

Thus, the upper limit on the expected event rate is weakened by a factor 4.1 due to spatial inhomogeneity.

This constraint assumes using $z_* = 50.2$ and, as we see from Figure 4-24, λ_s^+ grows as a function of a_*. Higher a_* produces a better prediction for λ_s^+ by accounting for more nonlinear clustering on small scales under gravity. As we see from the definition of λ_s^+ in equation (4.84), the value of λ_s^+ is mostly determined by voids since the sum in equation (4.84) is dominated by voids, for which each term in the series is higher and the total number of terms in the sum is greater. On the other hand

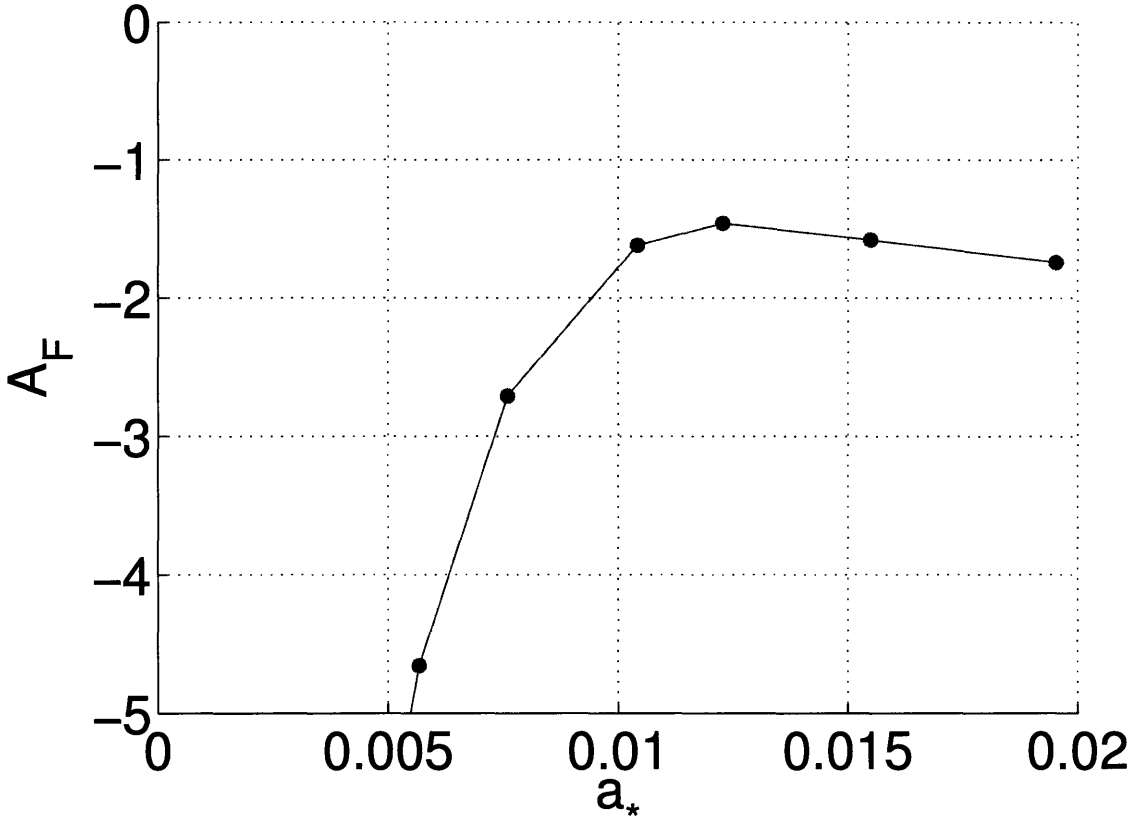

Figure 4-23: Flux probability distribution function slope defined by equation (4.87). The points on the curve shown in Figure 4-22 were used to define the slope.

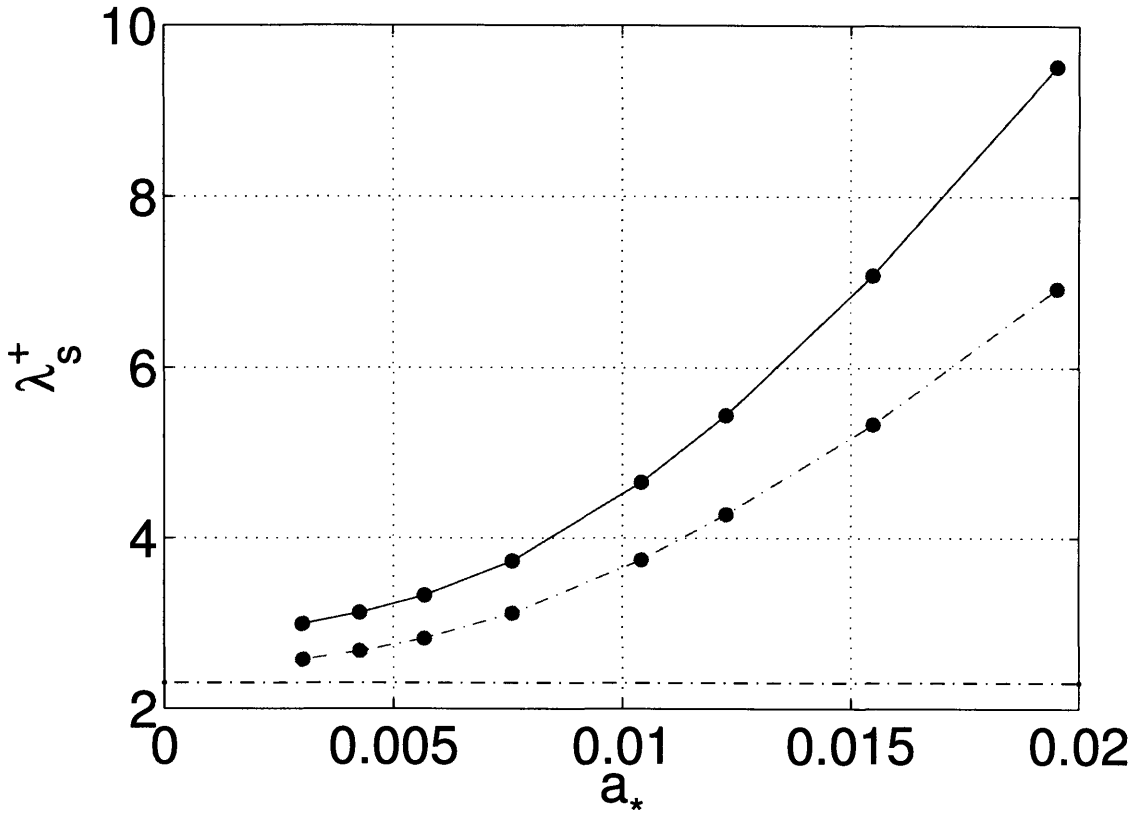

Figure 4-24: Upper solid line curve: The upper limit on the value of λ_0 at C.L. = 90% found from equation (4.83) as a function of expansion factor $a_* \equiv (1+z_*)^{-1}$ beyond which the clustering is assumed to be frozen. Lower solid line curve: same but with variations in velocities eliminated (see discussion further in the main text). The horizontal dash-dotted line shows the level of $\ln[1/(1-\text{C.L.})] \approx 2.303$.

the contribution of highly overdense cells into the sum is negligible due to the large ratio $(F_{\rm sym}(c)/F_0)$ for those cells resulting in relatively small overall contribution of overdense cells to the sum in equation (4.84). Since voids are becoming more and more underdense and their volume expands under nonlinear clustering it can be seen from equation (4.84) that λ_s^+ should indeed grow as it does in Figure 4-24.

If we were able to evolve the simulation all the way up to the present time without having to use frozen clustering model, we would find that since with the power spectrum slope $n = -3$ specified in Section 4.3.2 the perturbations of all scales are collapsing roughly at the same time, as soon as the perturbation of the length scale of our box size collapses, the voids will be filled with smaller halos that had previously collapsed onto and flown through the large scale halos. This will finally lead to a flattening of the curve in Figure 4-24. The level of this flattening can only be deduced from larger simulations than ours. However we can safely assume that the

rise of λ_s^+ as a function of the expansion factor is monotonic and therefore

$$\lambda_s^+(50.2) \leq \lambda_s^+(0) \,. \tag{4.90}$$

Using simulations of small scale structure we have therefore arrived to a correction on the upper limit of neutralino cross section given by [1]. Combining the above equation with equations (4.89) and (4.70) we now have a modified expression for the upper limit on the neutralino cross section that includes an account of small scale structure

$$\frac{\mathcal{E}\mathcal{A}}{M}\frac{\sigma_0}{m_\chi} \leq \frac{\lambda_s^+(z_*)}{\lambda_0^+}\frac{\ln(1/(1-\text{C.L.}))}{F_0} \tag{4.91}$$

which equals $4.1\ln(1/(1-\text{C.L.}))/F_0$ for $z_* = 50.2$. The upper limit by equation (4.91) is at least a factor of 4.1 weaker than the upper limit on the assumption of a uniform matter distribution on small scales.

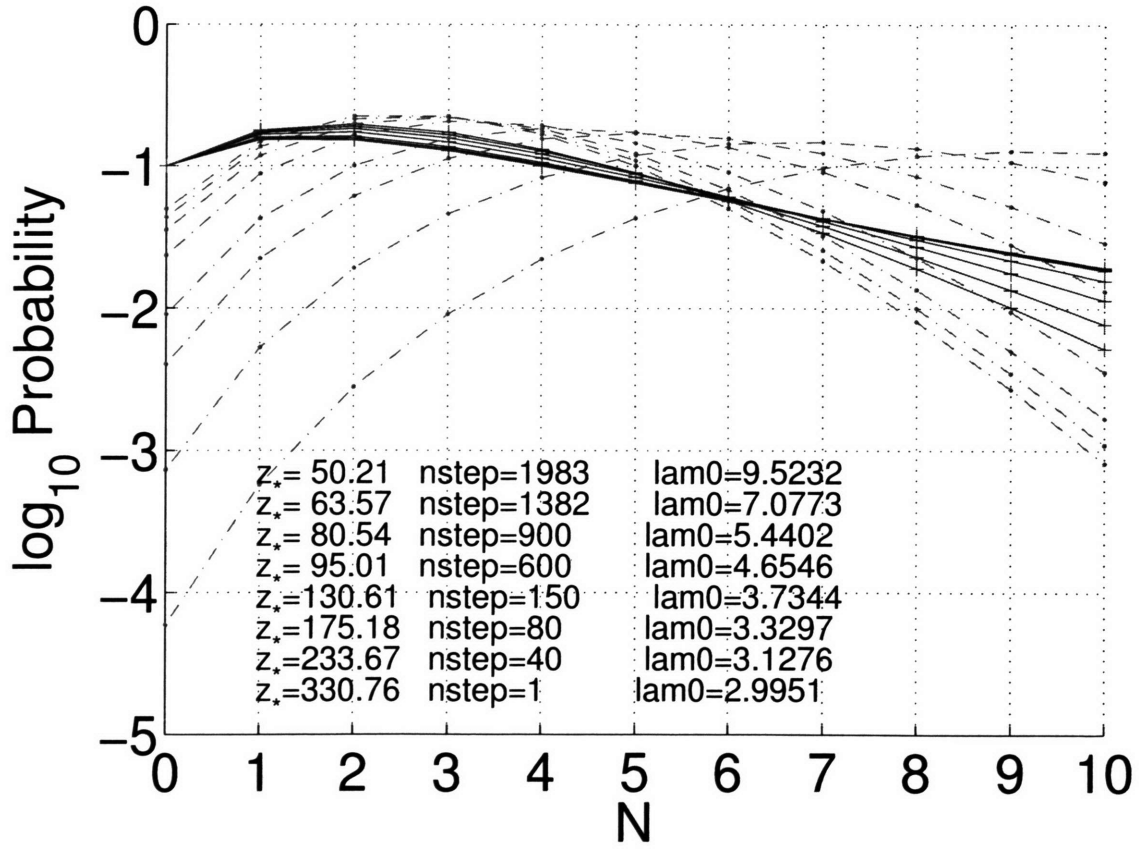

Figure 4-25: Solid lines: Probability distribution for detection of N neutralino collisions in the CDMS experiment given by equation (4.82) under assumption of the 90% of C.L. equation (4.92) for a selection of z_*. Curves become more flat with the increase of the expansion factor. Dash-dotted lines: Poisson distribution with Poisson average given by lam0=$\lambda_s^+(z_*)$.

In Figure 4-25 we present the probability to have N detections in an experiment, defined by equation (4.82) for our choices of z_*. We assumed the upper possible value of λ_0 within the confidence limit for each z_*

$$\lambda_0(z_*) = \lambda_s^+(z_*) \,, \tag{4.92}$$

which leads through equations (4.82) and (4.84) to $P(0) = 1 - \text{C.L.}$ for each a_*, as can be checked in the figure. For comparison we show the Poisson probability distribution $P_{\langle\lambda_{\rm cell}\rangle}(N)$ with the Poisson average

$$\langle\lambda_{\rm cell}\rangle = \frac{1}{N_{\rm c}} \sum_c \lambda_{\rm cell}(c, z_*)$$

at $z = 0$ thus representing the case of uniform matter distribution for the same average cell mass flux as the initial model.

In agreement with equation (4.91) the assumption of spatial uniformity leads to a lower probability of having no detections. On the other hand, if the experiment takes place at the same moment the detector passes through a dense clump of matter, the Poisson average [see equation (4.79)] of the number of detections in the experiment might greatly exceed unity in which case the probability of having many detections in one experiment sharply increases. Although the likelihood that the detector passes through a clump is small, overall the probability of having many detections is greater than in the spatially uniform case. Due to probability normalization, the pure Poisson case must have higher probability than the spatially inhomogeneous case for intermediate N.

It is interesting to test whether the variation of $\lambda_s^+ - \lambda_0^+$ is due to the spread in velocity or number density. In order to perform this test, we averaged the quantity $\langle v_{\rm sp}\rangle_c$ over all the cells in the simulation volume and use the resulting constant value instead of $\langle v_{\rm sp}\rangle_c$ in equation (4.81), therefore eliminating variations due to changes in the average particle velocity in cells. Since the mass flux in such a model is proportional to the local number density in Figure 4-26 we present the number density distribution function for comparison with Figure 4-22. We measured the self-similar region of the number density distribution function and found the slope $A_{\rm F} = -2.9$ as compared to equation (4.88). In Figure 4-24 (the lower dashed curve) we present the function $\lambda_s^+(a_*)$ computed using equation (4.81) with variations in velocities eliminated. The ratio of the variations $\lambda_s^+(a_*) - \lambda_0^+$ computed with and without velocity averaging tends asymptotically to a value close to 2/3 as the expansion factor increases. Thus, most of the variation in $\lambda_s^+ - \lambda_0^+$ is due to density inhomogeneity. The effect is easy to understand — particles in voids tend to have low velocities while particles have high velocities in virialized halos.

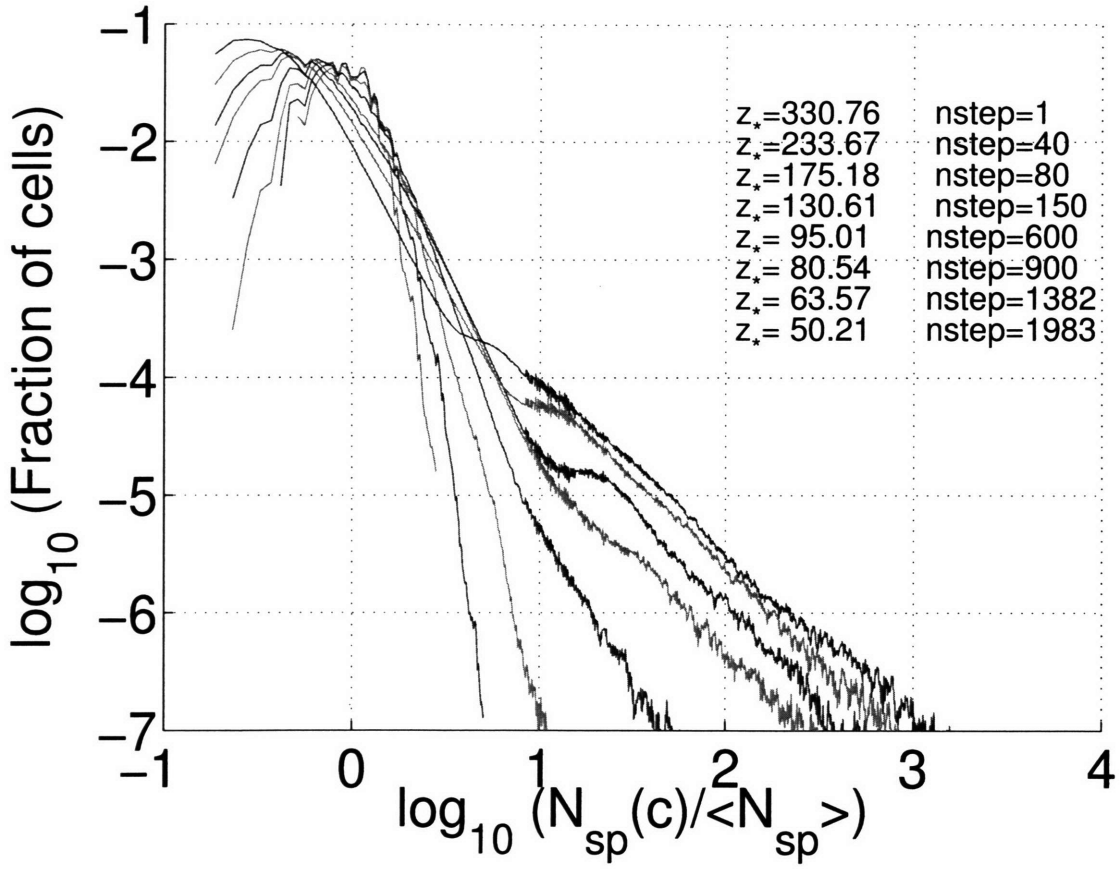

Figure 4-26: Number density distribution of cells in the simulation volume parametrized by z_*. The curve with the smallest range of densities corresponds to the highest z_*.

4.3.5 Effective Use of the Adaptive P^{3}M code for Caustic Simulation

In this chapter we used the adaptive P^{3}M code that we developed and described in Chapter 3. It took us 12 days to evolve the caustic simulation from redshift 350 to redshift 50.2 on 25 nodes of the same cluster of computers as described in Chapter 3. The dependence of the cumulative time to evolve the simulation up to a given expansion factor and the time per timestep on the expansion factor are shown in Figure 4-28.

It is very useful to apply the adaptive and load balancing technique developed in Chapters 2 and 3 for the short range force calculation. As we saw in Section 4.3.3, at the end of the simulation most particles belong to one of the few halos in the simulation box, thereby a small fraction of the Hilbert Curve mesh cells are highly occupied. The adaptive force technique is applied for force calculation for those cells, speeding up the short range force computation by orders of magnitude. At the end of the simulation run roughly 80% of all the wallclock time was spent computing the

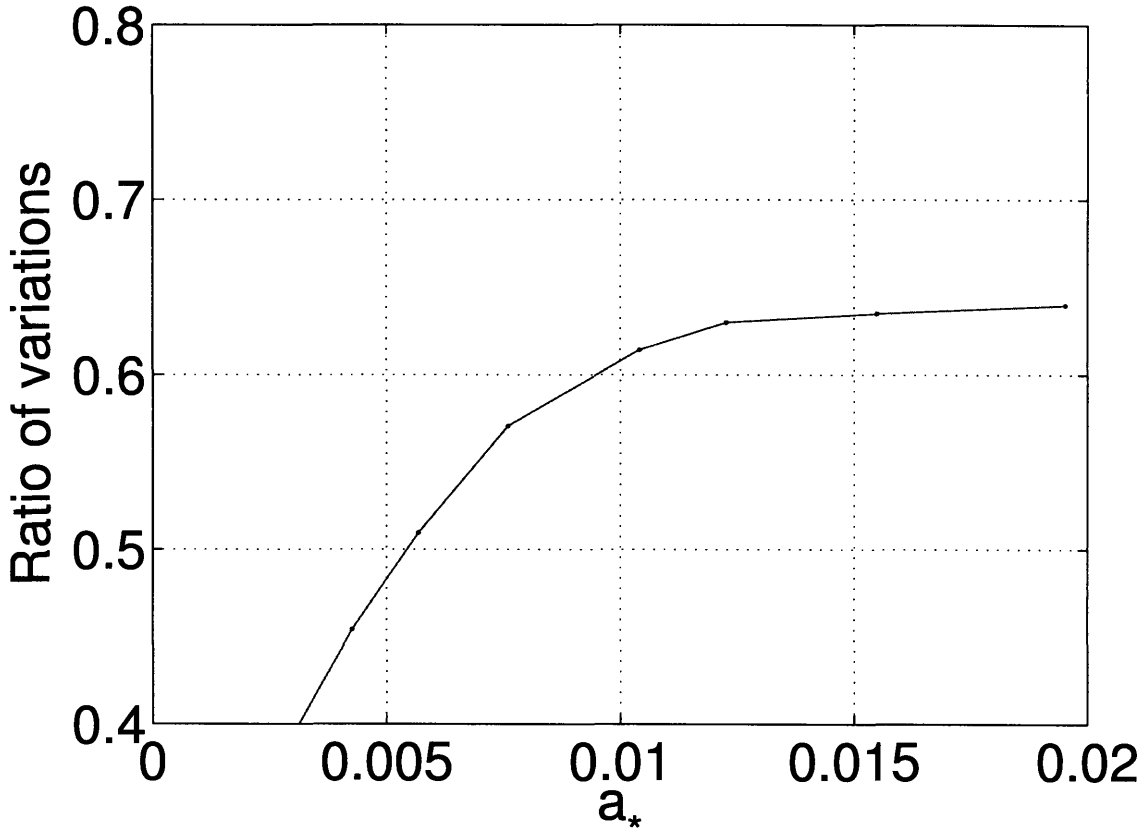

Figure 4-27: The ratio of the variations $\lambda_s^+(a_*) - \lambda_0^+$ computed with and without velocity averaging. Most of the variation is due to density effects, but about 1/3 of the variation is due to velocity effects.

forces adaptively for those few cells.

As shown on the left panel of Figure 4-29, the total number of mesh refined cells is only 0.25% of all the HC mesh cells. Nevertheless, at the end of the run, half of the 100 computing processed the cells whose refinement number is at or above 4. The computational work is distributed uniformly: most of the processes concentrate their work on those few cells whose computational load is immense. Figure 4-29 verifies that the load imbalance remains under 30% during the whole run.

We see, that the load imbalance reached 30% towards the end of the run, as compared to $\approx$ 12% for the non-adaptive P^{3}M run for ΛCDM described in Chapter I. The load imbalance is higher because we are using a different algorithm for computing the short range forces for heavily load cells and the distribution of matter is much more inhomogeneous on the scales comparable to the simulation volume, leading to a greater differences in tasks assigned to processes. The mesh refinement number of a cell, set by the workload continuity scheme given by equations (3.46) and (3.48) might significantly fluctuate with timesteps even if the number of computations needed to perform the adaptive force computation does not change. At the end of the run,

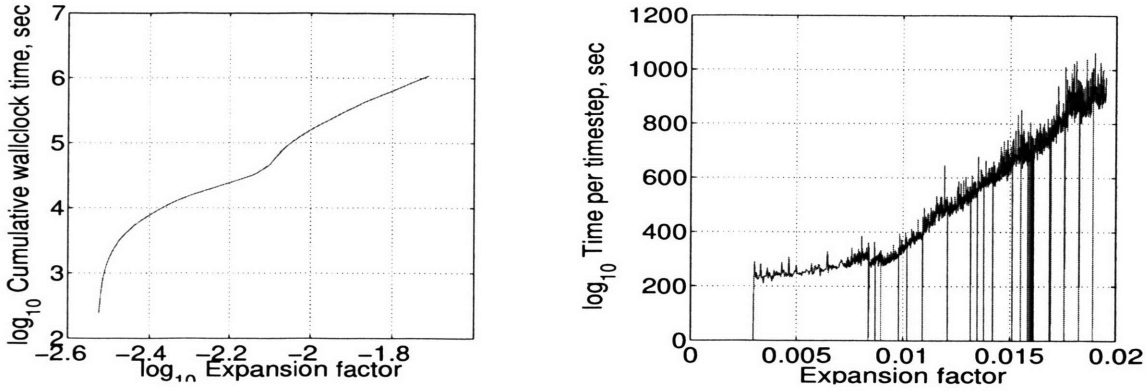

Figure 4-28: Left panel: Cumulative time to evolve the simulation to a given expansion factor as a function of expansion factor. This plot can be extrapolated to get the timing to evolve to a future expansion factor. Right panel: Time per timestep versus expansion factor.

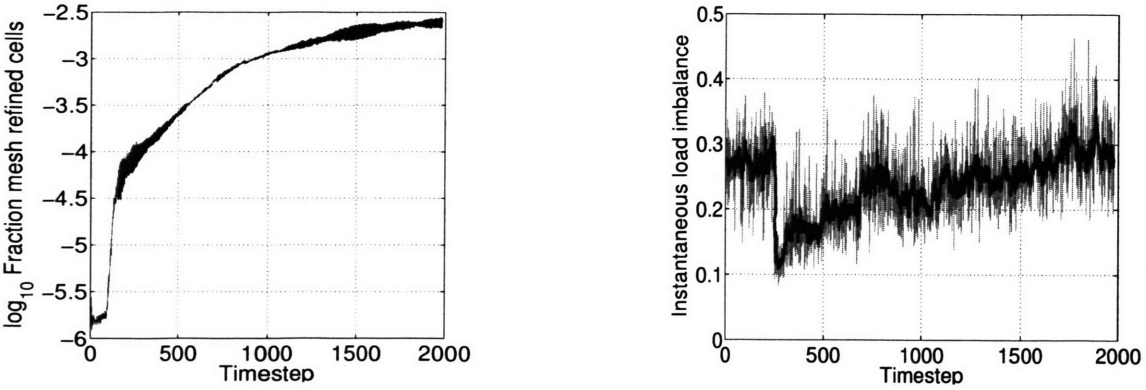

Figure 4-29: Left: Fraction of mesh refined cells of the total number of HC mesh cells. Right: Instantaneous load imbalance as a function of timestep.

the total number of mesh refined cells (whose mesh refinement number is non-zero) changes widely each timestep within the range between 5.2×10^4 and 6.6×10^4 cells (as can be deduced from the left panel of Figure 4-29). Such changes can be due to CPU fluctuations, inaccuracy of our scheme (3.48) or the numbers (3.50) being not optimal, all of these leading to the high uncertainty of the predicted cell workloads on the basis of the latest force calculation, or the breakdown of equation (2.29).

The load imbalance can be significantly reduced in the adaptive P^3M by putting a limitation on the rate (per timestep) at which a mesh refinement number changes for a cell within the simulation volume.

Bibliography

[1] Akerib, D. S., et al. 2003, Phys. Rev. D, 68, 082002

[2] Akerib, D. S., et al. 2004, Phys. Rev. Lett., 93, 211301

[3] Arnold, V. I. 1978, Mathematical Methods of Classical Mechanics (New York: Springer-Verlag)

[4] Bernabei, R., et al. 1996, Phys. Lett. B, 389, 757

[5] Bertschinger, E. 1985, Astrophys. J. Suppl., 58, 39

[6] Bertschinger, E. 1991, in After the First Three Minutes, ed. Holt, S., Trimble, V., & Bennett, C. (New York: AIP), 297

[7] Bertschinger, E. 1995, COSMICS software release (astro-ph/9506070)

[8] Bertschinger, E. 1996, in Cosmology and Large Scale Structure, proc. Les Houches Summer School, Session LX, ed. R. Schaeffer, J. Silk, M. Spiro, and J. Zinn-Justin (Amsterdam: Elsevier Science), 273

[9] Binney, J., & Tremaine, S. 1994, Galactic Dynamics (Princeton: Princeton University Press)

[10] Bode, P., & Ostriker, J. 2003, Astrophys. J. Suppl., 145, 1

[11] Bottino, A., Donato, F., Fornengo, N., & Scopel, S. 2003, Phys. Rev. D, 68, 043506

[12] Butz, A.R. 1971, IEEE Trans. Comp., 20, 424

[13] Couchman, H.M.P. 1991, Astrophys. J., 368, L23

[14] Davé, R., Dubinski, D.R., & Hernquist, L. 1997, New A., 2, 277

[15] Davis, M., & Peebles, P. J. E. 1983, Astrophys. J., 267, 465

[16] Diemand, J., Moore, B., & Stadel, J. 2005, Nature, 433

[17] Doroshkevich, A. G., & Shandarin, S. F. 1978, Mon. Not. R. Astron. Soc., 182, 27

[18] Dubinski, J. Kim, J., Park, C., & Humble, R. 2004, New A., 9, 111

[19] Ellis, J., Olive K., Santoso, Y. et al. 2003, Phys. Lett. B, 565, 176

[20] Efstathiou, G., & Eastwood, J.W. 1981, Mon. Not. R. Astron. Soc., 194, 503

[21] Efstathiou, G., Davis, M., Frenk, C.S., & White, S.D.M. 1985, Astrophys. J. Suppl., 57, 241

[22] Frigo, M., & Johnson, S., Fast Fourier Transform implementation at http://www.fftw.org/

[23] Ferrell, R., & Bertschinger, E. 1994, Int. J. Mod. Phys. C, 5, 933

[24] Ferrell, R., & Bertschinger, E. 1995, in proc. Soc. Comp. Sim. Multiconference (astro-ph/9503042)

[25] Fillmore, J., & Goldreich, P. 1984, Astrophys. J., 281, 1

[26] Fryxell, B. et al. 2000, Astrophys. J. Suppl., 131, 273; http://flash.uchicago.edu/

[27] Gates, E., Gyuk, G., & Turner, M. 1995, Astrophys. J., 449, L123

[28] Gelb, J.M., & Bertschinger, E. 1994, Astrophys. J., 436, 467

[29] Green, A., Hofmann, S., & Schwarz, D. 2004, Mon. Not. R. Astron. Soc., 353, L23

[30] Helmi, A., White, S. D. M., & Springel V. 2002, Phys. Rev. D, 66, 063502

[31] Hockney, R.W., & Eastwood, J.W. 1988, Computer Simulation Using Particles (Bristol: Adam Hilger)

[32] Hoffman, Y., & Shaham, J., 1985, Astrophys. J., 297, 16

[33] Jungman, G., Kamionkowski, M., & Griest, K. 1996, Phys. Rep., 267, 195

[34] Kinney, W. H., & Sikivie, P. 2000, Phys. Rev. D, 61, 087305

[35] Lewin, J., & Smith, P. 1996, Astroparticle Physics, 6, 87

[36] Ling, Fu-Sin, Sikivie, P., & Wick, S. 2004, Phys. Rev. D, 70, 123503

[37] MacFarland, T., Couchman, H.M.P., Pearce, F.R., & Pichlmeier, J. 1998, New A., 3, 687

[38] Merz, H., Pen, U.-L., & Trac, H. 2004, submitted to Mon. Not. R. Astron. Soc. (preprint astro-ph/0402443)

[39] Moore, D. 1994, Hilbert curve implementation at http://www.caam.rice.edu/%7Edougm/twiddle/Hilbert/

[40] Pilkington, J., & Baden, S. 1996, IEEE Trans. Par. Dist. Systems, 7, 288

[41] Plummer, H. C. 1911, Mon. Not. R. Astron. Soc., 71, 460

[42] Quinn, T., Katz, N., Stadel, J., & Lake, G. 1997, preprint (astro-ph/9710043)

[43] Ruth, R. D. 1983, IEEE Trans. Nucl. Sci., 30, 2669

[44] Salmon, J., & Warren, M. 1994, J. Comp. Phys., 111, 136

[45] Saha, P., & Tremaine, S. 1992, Astronomical. J., 104, 1633

[46] Schmid C., Schwarz D. J., & Widerin P. 1999, Phys. Rev. D, 59, 043517

[47] Sikivie, P. 1999, Phys. Rev. D, 60, 6

[48] Shandarin, S. F., & Zel'dovich Ya. B. 1989, Rev. Mod. Phys., 61, 185

[49] Springel, V., Yoshida, N., & White, S.D.M 2001, New A., 6, 79

[50] Tremaine, S. 1999, Mon. Not. R. Astron. Soc., 307, 877

[51] Waldsley, J.W., Stadel, J., & Quinn, T. 2004, New A., 9, 137

[52] Yoshida, H. 1990, Phys. Lett., A150, 262

[53] Zel'dovich, Ya., B. 1970, Astron. & Astrophys., 5, 84