Selected Methods in N-body Simulations

Aleksy Bałaziński

A thesis submitted in partial fulfillment of the requirements
for the degree of
Master of Science in Engineering
in **Applied Computer Science**

Faculty of Electrical Engineering
Warsaw University of Technology

June 7, 2025

This page intentionally left blank

Abstract

This thesis discusses the implementation and conducts a comparative analysis of three prominent algorithms used in gravitational N-body simulations: the particle-mesh (PM) method, the particle-particle particle-mesh (P^3M) method, and the Barnes-Hut algorithm. The N-body simulations are a crucial tool in astrophysical research where they are used for modeling systems where the gravitational interactions dominate. The primary goal of the work is to develop high-performance implementations of these methods from scratch and to evaluate their accuracy and computational efficiency. For the PM method, both CPU and GPU implementations are developed, with significant performance gains (up to 1200% speedup) observed for the GPU version. The short-range correction in the P^3M method is parallelized on the CPU leading to a four-fold speedup compared to a single-threaded version. The thesis also discusses a cache-friendly implementation of the octree construction procedure which offers a 40% speedup of tree construction and 15% speedup in the overall execution time of the Barnes-Hut algorithm. Through a series of test simulations, including spiral galaxies, globular clusters, and galaxy collisions, the methods are benchmarked for accuracy, computational cost, and scalability. The findings contribute practical insights into the strengths and limitations of each method.

Thesis supervisor: Maciej Twardy, PhD.

This page intentionally left blank

Streszczenie

Niniejsza praca przedstawia implementację oraz analizę porównawczą trzech ważnych algorytmów używanych w kontekście symulacji N-ciał; skoncentrowano się na metodach PM (ang. *Particle-mesh*), P^3M (ang. *Particle-particle particle-mesh*) oraz algorytmie Barnesa-Huta. Symulacje N-ciał odgrywają kluczową rolę w astrofizyce, gdzie używane są do modelowania systemów, w których dominującą siłą jest oddziaływanie grawitacyjne. Głównym celem pracy jest stworzenie autorskich wydajnych implementacji powyższych metod oraz zbadanie ich dokładności i wydajności obliczeniowej. Metoda PM została zaimplementowana zarówno w wersji na CPU, jak i GPU, przy czym wersja GPU osiągnęła znaczące przyspieszenie (nawet do 1200%). Korekta krótkozasięgowa w metodzie P^3M została zrównoleglona na CPU, co pozwoliło uzyskać czterokrotne przyspieszenie względem wersji jednowątkowej. W pracy omówiono równie implementację procedury konstrukcji drzewa oktalnego, która zapewnia 40% przyspieszenia budowy drzewa oraz 15% przyspieszenia całkowitego czasu wykonania algorytmu Barnesa-Huta dzięki efektywnemu wykorzystaniu pamięci podręcznej. Poprzez serię testowych symulacji, obejmujących modelowanie galaktyki spiralnej, gromady kulistej oraz kolizję dwu galaktyk, przeprowadzono ocenę dokładności, kosztu obliczeniowego oraz skalowalności metod. Wyniki pracy dostarczają praktycznych informacji na temat zalet i ograniczeń poszczególnych metod.

Promotor pracy: dr inż. Maciej Twardy.

This page intentionally left blank

Contents

Chapter 1

Introduction

The dominant force over large distances is the gravitational force. It controls the motion of planets in the solar system and is responsible for the evolution of stars and galaxies, and the whole cosmos [10]. Although the motion of two bodies orbiting each other lends itself to an analytical description, no general solution for systems comprising three or more bodies is known. Therefore, the importance of numerical simulations of many-body systems (usually referred to as *N-body simulations*) is hard to overestimate.

Even though the direct beneficiary of advances in the area of N-body simulations is the field of astrophysics, the development of such tools is highly interdisciplinary in nature. The theory behind algorithms that make fast simulations of this type possible is often involved mathematically and requires a good grasp of general physics. On the other hand, efficient implementation of these ideas demands proficiency in computer science, particularly in areas such as data structures, parallel computing, and performance optimization. Consequently, the development of N-body simulation codes is challenging but also intellectually rewarding.

1.1 Problem statement

To better understand the computational challenge before us, we begin by formulating the physical setup. The force exerted on a body with mass m_2 at the point $\mathbf{x}_2$ by a body with mass m_1 located at $\mathbf{x}_1$ can be expressed by the relation

$$\mathbf{F} = -G\frac{m_1 m_2}{|\mathbf{x}_{21}|^3}\mathbf{x}_{21} \tag{1.1}$$

where G is the gravitational constant $6.674 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ and $\mathbf{x}_{21} = \mathbf{x}_2 - \mathbf{x}_1$. Therefore, the evolution of a system of N bodies is governed by N equations

$$\ddot{\mathbf{x}}_i = -G\sum_{j\neq i}\frac{m_j}{|\mathbf{x}_{ij}|^3}\mathbf{x}_{ij}. \tag{1.2}$$

for each $i = 1, \ldots, N$. Direct application of Equation 1.2 is the basis of the so-called *particle-particle* method. The method is characterized by $O(N^2)$ time complexity (more precisely, it requires $(N-1)N/2$ operations if Newton's 3rd law is used in the computation). This running time characteristic $T(N)$ (running time of one simulation iteration) can be directly observed by running an implementation of

the method (see Figure 1.1). The exact form of $T(N)$ depends on the implementation details and

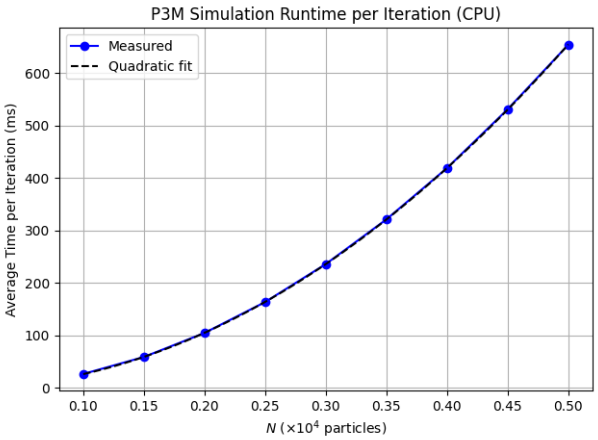


Figure 1.1: Scaling of the PP method.

the type of hardware used, but assuming that the code whose running time is shown in Figure 1.1 is not tragically inefficient, we can draw some general conclusions regarding the applicability of the PP method. Fitting a quadratic through the points shown in the figure reveals the following approximation: $T(1{,}000N) \approx 26.1N^2 + 0.32N$ [ms]. In a reasonable particle simulation, one often deals with tens of thousands of particles (if not more) and the simulation can span hundreds of iterations. Thus, assuming sensible values of $N = 50{,}000$ and 200 iterations, we get the expected running time of ≈ 3.5 hours! This clearly shows that more efficient algorithms have to be put in place.

1.2 Historical development

The *particle-mesh* (PM) technique, introduced by Roger W. Hockney and James W. Eastwood, was a significant improvement over the PP method. In the PM approach, the space is divided into a rectangular grid (or mesh) of cells. Each cell is assigned a portion of the mass of nearby particles, creating a density distribution $\rho(\mathbf{x})$. The relation between the density and gravitational potential ϕ, in the form of Poisson's equation

$$\nabla^2\phi = 4\pi G\rho, \tag{1.3}$$

is then used to obtain the potential at each cell center using Fourier techniques. The gravitational field $\mathbf{g}$ can then be calculated as $\mathbf{g} = -\nabla\phi$. Since $\mathbf{g}$ equals the acceleration due to gravity, we get $\ddot{\mathbf{x}}_i = \mathbf{g}(\mathbf{x}_i)$. The PM method and its theoretical grounding were most extensively described by Hockney and Eastwood in *Computer Simulation Using Particles* textbook published in 1988. However, one of the first papers treating the problem of solving the Poisson equation using Fourier analysis (*A Fast Direct Solution of Poisson's Equation Using Fourier Analysis* by R.W. Hockney) can be traced back to 1965.

The drawback of the PM method is its poor modeling of forces over short distances. Eastwood and Hockney also proposed a remedy for this problem in the form of a hybrid approach, called the *particle-particle-particle-mesh* method (or $\mathrm{P^3M}$ in short). In the $\mathrm{P^3M}$ method, the force on the i-th particle is

split into two components: *short-range* and *long-range* force. The long-range force is calculated using the PM method, whereas the short-range force can be found by direct summation of the forces due to nearby particles. The P^3M method was developed across the papers published by the authors between 1973 and 1980 with its thorough description given in *Computer Simulation Using Particles*.

The computational complexity of the PM and P^3M methods depends on the implementation of the potential solver used to calculate ϕ from Equation 1.3. For instance, if a fast Fourier transform is used, then the complexity of the PM algorithm is $O(N + N_g^3 \log N_g)$, where N_g is the number of cells in a single dimension of the grid (note it is linear in N). For the P^3M method, the worst-case scenario happens when all particles are clustered closely together, which causes the short-range $O(N^2)$ correction part to become dominant.

In 1986 Josh Barnes and Piet Hut introduced a hierarchical N-body simulation algorithm in the paper *A hierarchical O(N log N) force-calculation algorithm*, which is now known as the Barnes-Hut algorithm. The method developed by the authors uses a hierarchical subdivision of space into cubical cells where the subdivision is carried out recursively until at most one body can be found in any subdivision. The algorithm reduces the time complexity to $O(N \log N)$ by approximating groups of distant particles as a single "pseudo-particle".

Other notable examples of N-body simulation algorithms include (and are not restricted to) the Fast Multipole Method and Adaptive Mesh Refinement method [16].

1.3 Aim and scope

This thesis has three main goals:

1. Developing a program implementing the PM, P^3M, and Barnes-Hut algorithms focusing on correctness and performance,
2. Introducing these algorithms and analyze their strengths and weaknesses via error analysis and performance benchmarks using our implementation,
3. Implementing simple astronomical system models to test and evaluate the methods.

The field of N-body simulations has been extensively studied (see section 1.2), making novel contributions challenging. However, since our implementation is written from scratch, we aim to provide practical insights via:

1. Comparing GPU and CPU implementations of the PM method,
2. Parallelizing the short-range correction in the P^3M method,
3. Implementing a cache-friendly tree construction for the Barnes-Hut algorithm.

The full source code, including all algorithms and simulation models, is available at `https://github.com/AleksyBalazinski/ParticleSimulation`.

Chapter 2

Particle-mesh method

The particle-mesh method can be described as the following sequence of four steps:

1. Assign masses to mesh points,
2. Solve the field equation (Equation 1.3) on the mesh,
3. Calculate the field strength at mesh-points,
4. Find forces applied to individual particles by interpolation.

In this section, each of these steps will be described in more detail.

2.1 Mass assignment

The specifics of assigning mass from particles to mesh points depend on the density profile (or *shape*) associated with the particles. In general, the particles need not be represented as idealized dimensionless points; indeed, it is possible to construct a hierarchy of shapes where each successive member covers a larger number of mesh points and whose application leads to smaller numerical errors.

An infinite hierarchy of shapes with this property, as described by Hockney and Eastwood in [8], can be generated by successive convolutions with the "top-hat" function Π, defined as

$$\Pi(x) = \begin{cases} 1, & |x| < \frac{1}{2} \\ \frac{1}{2}, & |x| = 1 \\ 0, & \text{otherwise.} \end{cases}$$

The three most popular assignment schemes that hail from this family (and the ones implemented in our program) are the *nearest grid point* (NGP), *cloud in cell* (CIC), and *triangular shaped cloud* (TSC) schemes, with shapes S given by

$$S_{\text{NGP}} = \delta(x), \quad S_{\text{CIC}} = \delta(x) * \frac{1}{H}\Pi\left(\frac{x}{H}\right) = \frac{1}{H}\Pi\left(\frac{x}{H}\right), \quad S_{\text{TSC}} = \frac{1}{H}\Pi\left(\frac{x}{H}\right) * \frac{1}{H}\Pi\left(\frac{x}{H}\right) = \frac{1}{H}\Lambda\left(\frac{x}{H}\right),$$

where Λ is the triangle function

$$\Lambda(x) = \begin{cases} 1 - |x|, & |x| < 1 \\ 0, & \text{otherwise.} \end{cases}$$

For illustrative purposes, the shape S_{CIC} is depicted in Figure 2.1.

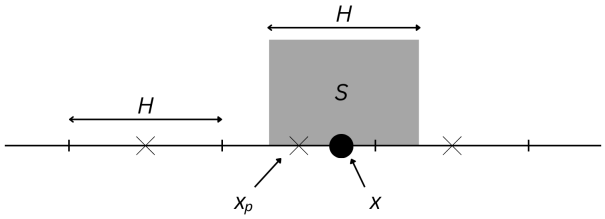


Figure 2.1: The CIC shape centered at x (particle position). The particle is within cell x_p; however, the cell x_{p+1} gets a non-zero density contribution from the particle.

In the one-dimensional case, the fraction of mass W_p assigned to mesh-point p from particle at position x is given by

$$W(x - x_p) = W_p(x) = \int_{x_p - H/2}^{x_p + H/2} S(x' - x)dx'.$$

A simple rule for relating the assignment function W defined above with shape S can be found by noticing that

$$W(x) = \int_{-H/2}^{H/2} S(x' - x)dx' = \int_{-\infty}^{\infty} \Pi\left(\frac{x'}{H}\right) S(x' - x)dx' = \Pi\left(\frac{x}{H}\right) * S(x).$$

This implies that

$$W_{\text{NGP}}(x) = \Pi\left(\frac{x}{H}\right), \quad W_{\text{CIC}}(x) = \Lambda\left(\frac{x}{H}\right), \quad W_{\text{TSC}}(x) = \Pi\left(\frac{x}{H}\right) * \frac{1}{H}\Lambda\left(\frac{x}{H}\right) = (\Pi * \Lambda)\left(\frac{x}{H}\right). \quad (2.1)$$

Splitting the domain of integration in the expression for W_{TSC} into five disjoint intervals shows that

$$(\Pi * \Lambda)(x) = \begin{cases} \frac{1}{8}(3 - 2|x|)^2, & \frac{1}{2} \le |x| < \frac{3}{2} \\ \frac{3}{4} - x^2, & |x| < \frac{1}{2} \\ 0, & \text{otherwise.} \end{cases}$$

Two- and three-dimensional versions of the assignment functions in Equation 2.1 are products of the assignment functions in each dimension. For example, the three-dimensional assignment function W is

$$W(\mathbf{x}) = W(x)W(y)W(z).$$

Hence, the mass assigned at mesh-point at $\mathbf{x_p}$ is

$$m(\mathbf{x_p}) = \sum_i m_i W_{\mathbf{p}}(\mathbf{x}_i),$$

or, in terms of density ρ,

$$\rho(\mathbf{x_p}) = \frac{1}{V} \sum_i m_i W_{\mathbf{p}}(\mathbf{x}_i), \quad (2.2)$$

where $V = H^3$ is the volume of a cell and i indexes the particles.

Algorithm 1 Density assignment algorithm

1: **for each** particle i **do**
2: **for each** cell $\mathbf{q}$ in $\mathcal{C}_S(\mathbf{x}_i)$ **do**
3: $\rho(\mathbf{x_q}) \leftarrow \rho(\mathbf{x_q}) + m_i W(\mathbf{x}_i - \mathbf{x_q})/V$

Obviously, Equation 2.2 is not suitable for direct application in the actual algorithm. Instead, we iterate over all particles, identify the parent cell $\mathbf{p}$ of each particle (and its neighborhood) and update ρ. This process is illustrated in Algorithm 1. The set $\mathcal{C}_S(\mathbf{x}_i)$ of cells that have to be considered while assigning density from the i-th particle depends on the shape S of the particle. Specifically, we have $\mathcal{C}_{\mathrm{NGP}}(\mathbf{x}) = \{[\mathbf{x}/H]\}$, $\mathcal{C}_{\mathrm{CIC}}(\mathbf{x}) = \{\lfloor \mathbf{x}/H \rfloor + \mathbf{t} \mid t_i = 0, 1\}$, and $\mathcal{C}_{\mathrm{TSC}}(\mathbf{x}) = \{[\mathbf{x}/H] + \mathbf{t} \mid t_i = -1, 0, 1\}$. It follows that $|\mathcal{C}_{\mathrm{NGP}}(\mathbf{x})| = 1$, $|\mathcal{C}_{\mathrm{CIC}}(\mathbf{x})| = 8$, and $|\mathcal{C}_{\mathrm{TSC}}(\mathbf{x})| = 27$ which illustrates the increasing computational cost resulting from using higher-order assignment schemes. We note that Algorithm 1 can be parallelized if atomic increments are used in line 3.

2.2 Solving the field equation

The Poisson equation (Equation 1.3) can be restated in integral form

$$\phi(\mathbf{x}) = \int \mathcal{G}(\mathbf{x} - \mathbf{x}')\rho(\mathbf{x}')dV',$$

which has the following discrete analogue

$$\phi(\mathbf{x_p}) = V \sum_{\mathbf{p}'} \mathcal{G}(\mathbf{x_p} - \mathbf{x_{p'}})\rho(\mathbf{x_{p'}}), \tag{2.3}$$

where $\mathcal{G}$ is the Green's function (potential due to unit mass). The right-hand side of Equation 2.3 is a convolution sum that runs over a finite set of mesh points. If we assume periodic boundary conditions, we can apply the discrete Fourier transform to both sides and use the convolution theorem to conclude that[1]

$$\hat{\phi}(\mathbf{k}) = \hat{\mathcal{G}}(\mathbf{k})\hat{\rho}(\mathbf{k}). \tag{2.5}$$

An approximation to $\hat{\mathcal{G}}$ can be found using a discretized version of the Laplacian in Equation 2.3.

[1] In this work, the Hockney & Eastwood definition of DFT is used, i.e.

$$D(x_p) = \frac{1}{L}\sum_{l=0}^{N-1} \hat{D}(k)e^{ikx_p}, \quad \hat{D}(k) = H\sum_{p=0}^{N-1} D(x_p)e^{-ikx_p},$$

where $x_p = pH$. The conversion between this form and another popular definition,

$$\widetilde{D_H}(k) = \sum_{p=0}^{N-1} D_H(p)e^{-i2\pi kp/N}, \tag{2.4}$$

is given by

$$\widetilde{D_H}(k) = \frac{1}{H}\hat{D}\left(\frac{2\pi}{NH}k\right),$$

where $D_H(p) = D(pH)$.

Specifically, for a 7-point stencil,

$$\begin{aligned}4\pi G\rho(\mathbf{x}_{ijk}) = &\frac{\phi(\mathbf{x}_{i-1,j,k}) - 2\phi(\mathbf{x}_{ijk}) + \phi(\mathbf{x}_{i+1,j,k})}{H^2} \\ &+ \frac{\phi(\mathbf{x}_{i,j-1,k}) - 2\phi(\mathbf{x}_{ijk}) + \phi(\mathbf{x}_{i,j+1,k})}{H^2} \cdot \\ &+ \frac{\phi(\mathbf{x}_{i,j,k-1}) - 2\phi(\mathbf{x}_{ijk}) + \phi(\mathbf{x}_{i,j,k+1})}{H^2}.\end{aligned}$$

Applying the discrete Fourier transform to both sides and using the shift theorem, we get

$$\begin{aligned}4\pi G\hat{\rho}(\mathbf{k}) &= \frac{1}{H^2}\sum_{i=1}^{3}\left(e^{-iHk_i} + e^{iHk_i} - 2\right)\hat{\phi}(\mathbf{k}) \\ &= \frac{1}{H^2}\sum_{i=1}^{3}\left(e^{iHk_i/2} - e^{-iHk_i/2}\right)^2\hat{\phi}(\mathbf{k}) \\ &= -\frac{4}{H^2}\sum_{i=1}^{3}\sin^2\left(\frac{Hk_i}{2}\right)\hat{\phi}(\mathbf{k}).\end{aligned}$$

and hence

$$\hat{\phi}(\mathbf{k}) = -4\pi G\underbrace{\frac{(H/2)^2}{\sin^2(Hk_1/2) + \sin^2(Hk_2/2) + \sin^2(Hk_3/2)}}_{\hat{\mathcal{G}}(\mathbf{k})}\hat{\rho}(\mathbf{k}), \tag{2.6}$$

where $\hat{\mathcal{G}}$ can be identified by comparison with Equation 2.5. It is worth noting that the constant multiplier $(-4\pi G)$ is often left out of $\hat{\mathcal{G}}$ (this is the convention used in [8]). In the implementation, values of $\hat{\mathcal{G}}$ are computed only once and saved for future look-up.

In the one-dimensional case, the above discussion can be easily rephrased in terms of a diagonalization problem [5]. In one dimension, the Poisson equation is

$$\frac{d^2\phi}{dx^2} = \rho. \tag{2.7}$$

The interval $[a, b]$ on which we wish to find ϕ is assumed to be discretized into N points $x_j = a + jH$, where $H = (b-a)/(N-1)$ and $j = 0, \ldots, N-1$. Furthermore, we assume periodic boundary conditions so that $\phi(x_{-1}) = \phi(x_{N-1})$ and $\phi(x_N) = \phi(x_0)$. The approximation of Δ at x_j is

$$(\Delta_H\phi)_j \equiv \frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{H^2}.$$

Thus, the discrete version of Equation 2.7 reads $(\Delta_H\phi)_j = \rho_j$ or

$$\frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{H^2} = \rho_j,$$

where $\rho_j = \rho(x_j)$. This gives a system of N equations (one per each sampled value of ρ) with N unknowns (values of ϕ) of the following form

$$\underbrace{\frac{1}{H^2}\begin{bmatrix} 2 & -1 & & & -1 \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ -1 & & & -1 & 2 \end{bmatrix}}_{\Delta_H}\underbrace{\begin{bmatrix}\phi_0 \\ \phi_1 \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1}\end{bmatrix}}_{\vec{\phi}} = -\underbrace{\begin{bmatrix}\rho_0 \\ \rho_1 \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1}\end{bmatrix}}_{\vec{\rho}}. \tag{2.8}$$

Next we define $\bar{\psi}(k) \in \mathbb{C}^N$ by $\psi_j(k) \equiv e^{i2\pi jk/N} = \omega^{jk}$ for $j = 0, \ldots, N-1$. The j-th row of $H^2\Delta_H\bar{\psi}$ equals

$$\begin{aligned}(H^2\Delta_H\bar{\psi})_j &= -\psi_{j-1} + 2\psi_j - \psi_{j+1} = -\omega^{(j-1)k} + 2\omega^{jk} - \omega^{(j+1)k} \\ &= -\omega^{jk}(\omega^{-k} - 2 + \omega^k) = -\omega^{jk}(\omega^{k/2} - \omega^{-k/2})^2 \\ &= 4\omega^{jk}\sin^2\left(\frac{\pi k}{N}\right).\end{aligned}$$

Hence we have $(H^2\Delta_H\bar{\psi})_j = 4\sin^2(\pi k/N)\psi_j$ which implies

$$\Delta_H\bar{\psi}(k) = \frac{4}{H^2}\sin^2\left(\frac{\pi k}{N}\right)\bar{\psi}(k).$$

This results tells us that for any value of k, $\bar{\psi}(k)$ is an eigenvector of Δ_H with eigenvalue $(4/H^2)\sin^2(\pi k/N)$. For $k = 0, \ldots, N-1$ we get N linearly independent eigenvectors $\{\bar{\psi}(k)\}$ which necessarily form the basis of $\mathbb{C}^N$. This fact implies that Δ_H is diagonalizable. The matrix F of its eigenvectors is given by $F_{jk} = \omega^{jk}$, and the inverse is easily verified to be $F^{-1} = (1/N)F^\dagger$. The eigendecomposition of Δ_H is therefore

$$\Delta_H = F\Lambda F^{-1},$$

where $\Lambda = \mathrm{diag}((4/H^2)\sin^2(\pi k/N))$. Substitution into Equation 2.8 yields $F\Lambda F^{-1}\bar{\phi} = -\bar{\rho}$ or

$$F^{-1}\bar{\phi} = -\Lambda^{-1}F^{-1}\bar{\rho}.$$

Evaluating the left-hand side leads to the conclusion that $F^{-1}\bar{\phi}$ is nothing else but the DFT of ϕ. Indeed,

$$(F^{-1}\phi)_k = \frac{1}{N}(F^\dagger\phi)_k = \frac{1}{N}\sum_{j=0}^{N-1}F^\dagger_{kj}\phi_j = \frac{1}{N}\sum_{j=0}^{N-1}\omega^{-kj}\phi_j = \frac{1}{N}\sum_{j=0}^{N-1}e^{-2\pi ijk/N}\phi_j$$

which is exactly the discrete Fourier transform of ϕ (using the standard definition of the DFT). Thus, we see that the DFT of the Green's function derived in Equation 2.6 (or rather the one-dimensional variant thereof) is the k-th eigenfunction of the discretized Laplacian (with slight differences due to a different definition of the DFT being used there). By following the line of reasoning presented above, we observe a deep connection between the DFT and the eigendecomposition of the discretized Laplace operator.

A natural question that arises in the context of this discussion is whether a similar relation holds in the continuous case. The answer turns out to be positive; assuming free-space boundary conditions, the complex exponential $e^{2\pi i\mathbf{k}\cdot\mathbf{x}}$ (kernel of the inverse Fourier transform) is an eigenfunction of the Laplace operator (with eigenvalue $-(2\pi k)^2$) and the Fourier transform allows for a "diagonalization" of the Laplacian [5]. More precisely, we have

$$\Delta = F\Lambda F^{-1},$$

where F is the inverse FT, and $\Lambda = -(2\pi k)^2$. Applying this result to the Poisson equation yields $F^{-1}\phi = \Lambda^{-1}F^{-1}\rho$ or

$$\hat{\phi} = -\frac{1}{4\pi^2k^2}\hat{\rho}, \tag{2.9}$$

where the circumflex denotes the standard Fourier transform and $k = |\mathbf{k}|$. The function $-(2\pi k)^2$ is sometimes used instead of the Green's function $\mathcal{G}$ defined in Equation 2.6. This approach is the basis of

the "poor man's Poisson solver" (as dubbed by [8]). While implementing it, we have to take into account that in the standard definition of the DFT, nonnegative frequencies correspond to $0 \leq k \leq N/2$ and negative frequencies correspond to $N/2 + 1 \leq k \leq N - 1$ (see [12], pp. 607–608). This necessitates the following "index wrapping":

$$k_i = \begin{cases} i & \text{if } i \leq \frac{N}{2} \\ i - N & \text{if } i > \frac{N}{2} \end{cases},$$

where i is the index of the gridpoint.

2.3 Field strength calculation

The strength $\mathbf{g}$ of the gravitational field at mesh-point $\mathbf{x_p}$ can be approximated using a central difference. Our implementation currently supports two types of finite differences, described below.

The two-point finite difference operator $\mathbf{D}$, whose x component is given by

$$D_x(\phi)(\mathbf{x_p}) = \frac{\phi(\mathbf{x}_{i+1,j,k}) - \phi(\mathbf{x}_{i-1,j,k})}{2H}$$

(and analogously for the y and z components), is second order accurate.

The fourth-order accurate finite difference is given by

$$D_x(\phi)(\mathbf{x_p}) = \alpha \frac{\phi(\mathbf{x}_{i+1,j,k}) - \phi(\mathbf{x}_{i-1,j,k})}{2H} + (1 - \alpha) \frac{\phi(\mathbf{x}_{i+2,j,k}) - \phi(\mathbf{x}_{i-2,j,k})}{4H},$$

where $\alpha = 4/3$.

The difference between the accuracy of both methods is illustrated in Figure 2.2. The figure also provides insight into how the error depends on the value of parameter α.

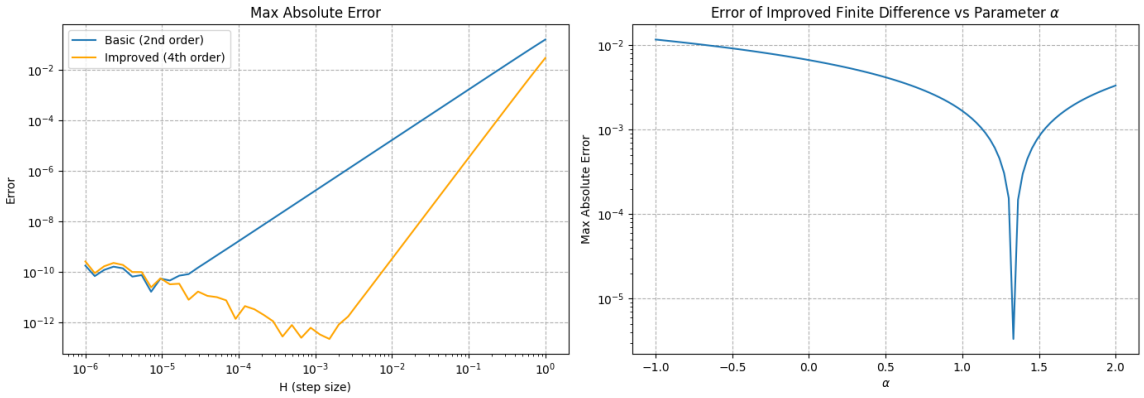


Figure 2.2: Left pane: Approximation error for second-order and fourth-order schemes. For small values of H, round-off errors dominate. Right pane: Approximation error vs. α in the improved finite difference scheme ($H = 0.1$). Note that the scheme is fourth-order accurate only for $\alpha = 4/3$ (cusp in the graph).

We can alternatively define the finite difference operators in terms of the delta function to get rid of the dependence on the differenced function. (Technically, the resulting quantities are functions rather than operators.) Consider, for example, the two-point finite difference in Equation 2.10. The definition

can be generalized beyond mesh points by letting

$$D_j(\phi)(\mathbf{x}) = -\frac{\phi(\mathbf{x}+H\mathbf{e}_j)-\phi(\mathbf{x}-H\mathbf{e}_j)}{2H} = -\int\left[\frac{\delta(\mathbf{x}+H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-H\mathbf{e}_j-\mathbf{x}')}{2H}\right]\phi(\mathbf{x}')d\mathbf{x}',$$

where $\mathbf{e}_j$ is the j-th standard basis vector. This generalization motivates us to define

$$D_j(\mathbf{x}) = \frac{\delta(\mathbf{x}+H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-H\mathbf{e}_j-\mathbf{x}')}{2H} \tag{2.10}$$

and

$$D_j(\mathbf{x}) = \frac{\delta(\mathbf{x}+H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-H\mathbf{e}_j-\mathbf{x}')}{2H} + (1-\alpha)\frac{\delta(\mathbf{x}+2H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-2H\mathbf{e}_j-\mathbf{x}')}{4H} \tag{2.11}$$

as the two-point and four-point finite difference operators, respectively.

If ϕ denotes the gravitational potential, then the field $\mathbf{g}$ is approximated at mesh point $\mathbf{x_p}$ as

$$\mathbf{g}(\mathbf{x_p}) = -\mathbf{D}(\phi)(\mathbf{x_p}).$$

2.4 Interpolation

The value of the field strength $\mathbf{g}(\mathbf{x})$ at the position particle's position $\mathbf{x}$ is calculated by interpolating the values of $\mathbf{g}$ from the neighboring mesh-points. Formally,

$$\mathbf{g}(\mathbf{x}) = \sum_{\mathbf{p}} W(\mathbf{x}-\mathbf{x_p})\mathbf{g}(\mathbf{x_p}).$$

In practice, there is no need to sum over all mesh points. Instead, we use an algorithm analogous to Algorithm 1 to only include the cells with non-zero contribution to the sum. The method is illustrated in Algorithm 2. It is important to note that to retain correct physical behavior, the interpolation and mass

Algorithm 2 Field strength interpolation

```
1: for each particle i do
2:     for each cell q in C_S(x_i) do
3:         g(x_i) ← Σ_q W(x_i − x_q) g(x_q)
```

assignment schemes must use the same shape to represent the particles. The procedure in Algorithm 2 is trivially parallelized by converting the sequential loop into a parallel one.

The procedures of density assignment and interpolation presented in Algorithm 1 and Algorithm 2 are high-level descriptions. More concrete formulations suitable for direct use in implementation are given in [8] and [9].

2.5 Code units

Implementation of the PM (and P^3M) methods can be simplified by switching to a system of dimensionless units, often called *code units*. The natural units of time and length in a PM simulation are H and

DT, respectively. Hence, length in a PM code is conveniently expressed in terms of multiples of H, and similarly, time intervals are given as a multiple of DT, i.e., the conversion relations are

$$x' = \frac{x}{H} \quad \text{and} \quad t' = \frac{t}{DT}.$$

From there, it follows that

$$v' = \frac{DT}{H}v \quad \text{and} \quad a' = \frac{DT^2}{H}a.$$

The expected relation $\mathbf{g}' = -\nabla'\phi'$ leads to the definition $\phi' = (DT^2/H^2)\phi$. By stipulating that we have $\nabla'^2\phi = \rho'$, we get $\rho' = DT^2 \cdot 4\pi G\rho$, $m' = (DT^2 \cdot 4\pi G/H^3)m$, and $G' = 1/(4\pi)$.

2.6 Properties of the calculated field

The accuracy of a simulation using a PM code depends on several factors. The user can decide to use any combination of the following elements that parameterize the program:

- mass assignment and interpolation scheme: NGP, CIC, TSC, or possibly any other scheme from the infinite hierarchy described in section 2.1;
- convolution with the Green's function derived from the discretized Laplacian (Equation 2.6) or the "poor man's solver" (Equation 2.9);
- gradient approximation: two-point or four-point finite difference;
- grid resolution: number of meshpoints in each dimension.

In this subsection, we analyze these choices from two different angles. We first focus on the properties of the field produced by a single source, which provides insight into the behavior of the simulation based on pair-wise interparticle interactions. Then, we analyze the global error in force calculation by comparing the PM-calculated forces acting on all particles in a typical simulation with forces produced by the PP method.

2.6.1 Local picture

The field produced by the PM method is neither homogenous nor isotropic. Anisotropy can be observed by measuring the field generated by a particle in two different directions. In Figure 2.3, the field strength calculated using the PM method (with the Green's function derived from discrete Laplacian) due to a single source at $x = H$ is shown in two variants: when measured along the x-axis (blue line) and the $x = y$ line (orange line). The difference between these two graphs illustrates the anisotropy of the calculated field. The figure also shows the field strength measured along the x-axis when the source was shifted by $H/2$ in the $-x$ direction (green line). Its deviation from the case when the source was placed at $x = H$ (the blue graph) exemplifies the inhomogeneity of the field computed using the PM method. As expected, the PM-calculated approximation gets better with increasing distance from the source; the inverse-square law is reproduced accurately for $r \gtrsim 4H$.

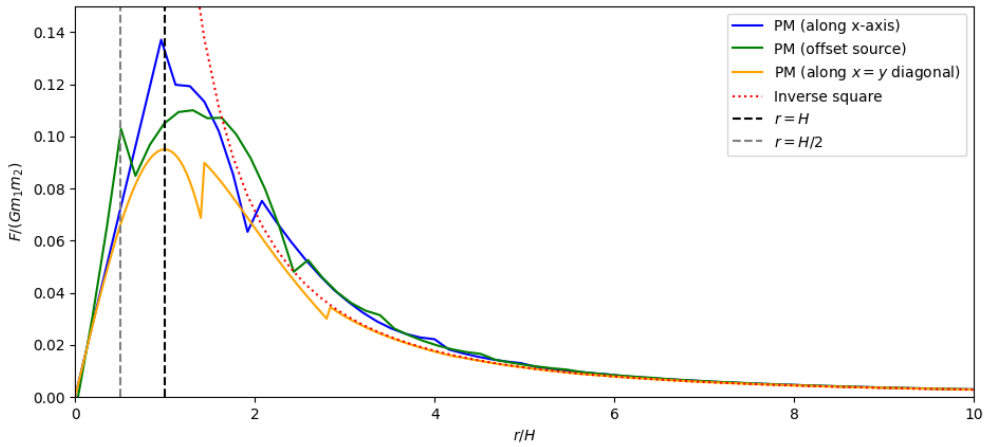
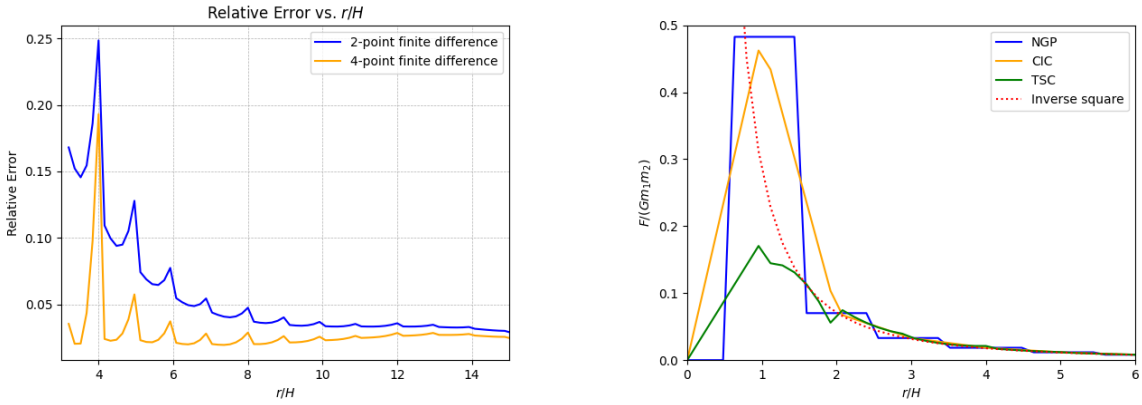


Figure 2.3: Anisotropy and inhomogeneity of the field as calculated by the PM method (TSC assignment, second order finite difference).

The single-source case analysis can be extended by considering the effect of choice of finite difference and mass assignment schemes on the relative error between the actual and approximated field strength. Figure 2.4 shows a comparison of (a) the relative error as a function of distance from the source for PM with second-order and fourth-order finite differences and (b) the field strength computed using different mass assignment schemes discussed in section 2.1. The conclusion that can be drawn from Figure 2.4 is



(a) Relative error of field strength approximation in different PM variants.

(b) Field strength calculated with the PM method using different assignment schemes.

Figure 2.4: Comparison of PM method performance. (a) Relative error due to finite difference accuracy. (b) Effect of mass assignment schemes on computed field strength (4-point finite difference).

that, as expected, the TSC mass assignment scheme and the four-point finite difference yield the best results.

The field produced by the PM method using the "poor man's" potential solver is similar to the variant employing the discretized Laplacian Green's function described above.

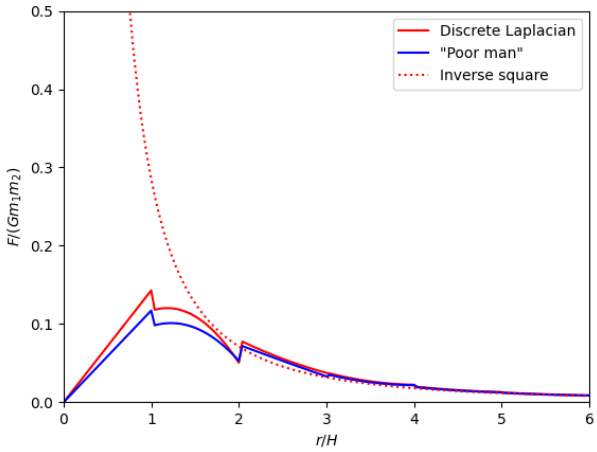


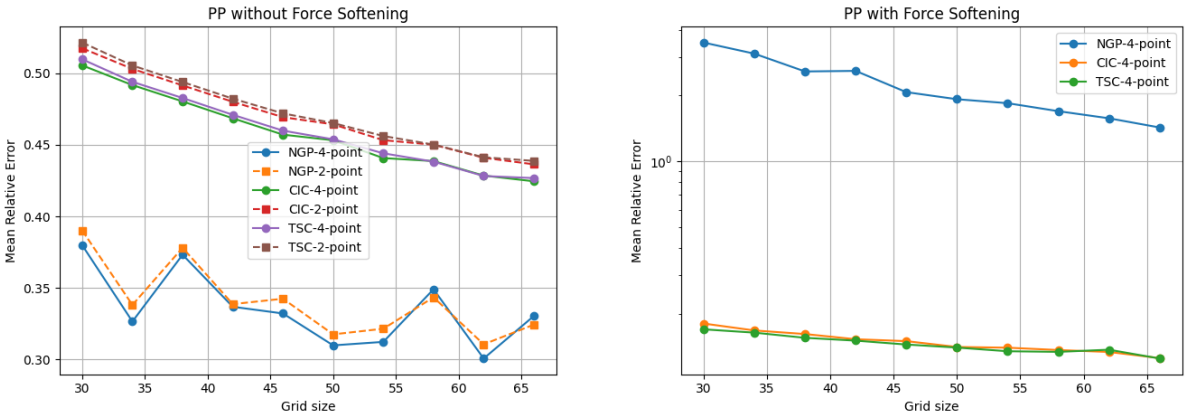
Figure 2.5: Field obtained potential calculated by the "Poor man's Poisson solver" vs. when using Equation 2.6.

2.6.2 Global picture

To calculate the global approximation error of a PM simulation, we calculated the average of relative differences over all particles ($N = 10{,}000$ in this case) between the forces produced by the PM method and the exact result obtained using the PP method. More precisely, we the error was defined as

$$\frac{1}{N}\sum_i \frac{|\mathbf{F}_i^{\text{approx.}} - \mathbf{F}_i^{\text{PP}}|}{|\mathbf{F}_i^{\text{PP}}|}. \tag{2.12}$$

The result for different grid resolutions is shown in Figure 2.6 ($30 \leq N_g \leq 70$). For this test, the particle positions were generated according to the uniformly decreasing density disk model. As can be seen in



(a) PP calculation without force softening.

(b) PP calculation with force softening.

Figure 2.6: Average force approximation error in the PM method (discrete Laplacian solver).

Figure 2.6a, for each of the assignment schemes, the error is minimized for the four-point difference, as expected. A more surprising result is that the errors are most significant for the TSC assignment scheme. This paradox is easily explained by considering the graphs in Figure 2.4b. As can be seen there,

the TSC assignment scheme gives rise to a force that accurately reproduces the inverse-square law force, but at the same time, it severely underestimates the force close to the source. This behavior is expected since it stems from the mass of any individual particle being spread between multiple cells. The CIC and NGP schemes, on the other hand, do not reproduce the actual force with the same level of accuracy, but the underestimation close to the source is not as severe as in the case of the TSC scheme. In some applications, for instance, in the context of a galaxy simulation, this is a favorable feature of the TSC scheme. The number of stars in a galaxy can be in the order of a trillion [20], whereas the number of particles in our tests is in the order of tens of thousands, giving an average of around $10^{12}/10^4 = 10^8$ of stars per particle. As can be seen, it would be a grave mistake to model the force due to a single particle by the force due to a dimensionless point mass. In such a case, it is a standard practice to use a softened gravitational force instead of the one given in Equation 1.1 [3]. By using the TSC scheme, force softening is automatically included in the calculation. It can be argued that obtaining the smoothness properties enjoyed by the higher-order assignment schemes should be treated as tangential to softening the force. It is for this reason that Hockney and Eastwood introduce the notion of *force sharpening* as an additional step in the PM method in [8]. In this modified version of the PM method, the Poisson equation takes the form

$$\nabla^2 \phi_p = \rho_p^*,$$

where ρ_p^* is obtained from the density ρ_p by solving the following system

$$1 + \frac{\rho_{p+1}^* - 2\rho_p^* + \rho_{p-1}^*}{8} = \rho_p,$$

where p indexes the mesh points. In this work, we do not investigate this matter further, nor do we implement this modification in our program.

Now we return to the test whose results are presented in Figure 2.6. In the light of the above discussion, it is more reasonable to compare the quality of the results of the PM method with a *softened gravitational force* defined in [22] as

$$\mathbf{F}_{ij}^{\text{soft}} = -G\frac{m_i m_j}{(r_{ij}^2 + \epsilon^2)^{3/2}}\mathbf{r}_{ij}. \tag{2.13}$$

The relative difference between the softened forces and the PM-calculated ones is shown in Figure 2.6b. The *softening length* in the PP method was set at $\epsilon = H$ in each run of the test so that it roughly matches the softening "induced" by the PM method (cf. Figure 2.4b).

2.7 Implementation

2.7.1 CPU variant

The most computationally involved step of the PM method is the potential solver. As explained in section 2.2, in our implementation, we solve the Poisson equation for the potential using the convolution method. This approach involves taking the DFT of the grid-defined density and later the inverse DFT of the product of transformed Green's function and the transformed density. Since the number of grid

points in a typical grid can be of the order of a million, choosing a fast enough implementation of the FFT is of paramount importance. We attempted to incorporate a handful of popular C++ FFT libraries into the program, and out of those considered, FFTW (`https://www.fftw.org/`) and kissfft (`https://github.com/mborgerding/kissfft`) provided the best performance. The running times for FFT and inverse FFT stages of the algorithm using each of the libraries are shown in Figure 2.1. As we

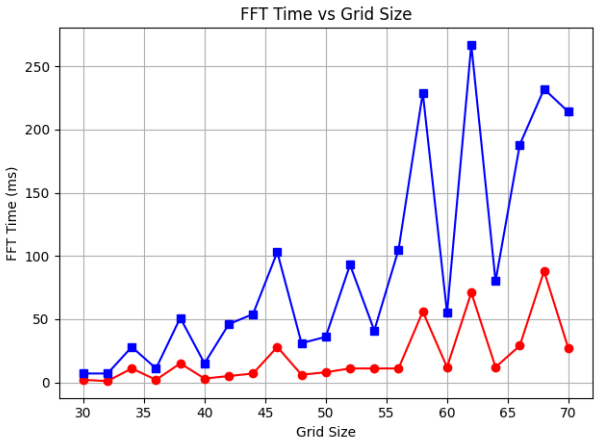


Figure 2.7: Time needed to perform the DFT and inverse DFT using kissft and FFTW libraries. The total grid size in the test was $(2N_g) \times (2N_g) \times N_g$, where N_g is labeled as "Grid Size" on the horizontal axis in the graph.

can see, FFTW considerably outperformed kissfft, with the former achieving an average five-fold speedup over the latter.

As noted in section 2.1, parallelization of the mass assignment step, which involves "spreading" the mass due to a particle over nearby cells, requires synchronization to avoid data races. The above requirement can be fulfilled by using mutexes or atomic operations. Traditionally, atomic RMW operations were supported by the C++ standard library only for the integer types; however, recently, C++20 introduced atomic RMW operations on floating point numbers (see `https://en.cppreference.com/w/cpp/atomic/atomic.html`). We conducted a simple benchmark on Intel Core i7-9750H CPU (three threads concurrently incrementing a global variable) to compare the performance of both approaches. The benchmark indicated that guarding the critical section with a mutex was around two times faster than using atomic operations. However, the same test run on a system with a different processor indicated that using an atomic counter offers better performance, suggesting that more careful benchmarking would be needed to reach a definitive conclusion.

2.7.2 GPU variant

As explained before, the mass assignment, field calculation at mesh points, and interpolation steps of the PM method are embarrassingly parallel. For this reason, rewriting the PM code for the GPU does not pose much difficulty. In our implementation, we used CUDA C++ language extension to implement the method on the GPU. A thorough introduction to the arcana of programming NVIDIA GPU devices

is given in [11]. In this work, we only remark on the aspects directly related to our use case.

The memory hierarchy of a GPU (also referred to as the *device*) is organized into three main levels:

1. global memory (accessible by all threads as well as *host* (CPU); high latency),
2. shared memory (accessible by threads in the same threadblock typically up to 48 KB; very low latency),
3. local storage (per thread storage in the form of registers managed by the compiler).

One step of the PM method that could potentially benefit from transferring data to shared memory is the gradient calculation step. For example, if the 2-point finite difference approximation to the Laplacian is used, the potential from six cells needs to be fetched from memory. At the same time, there is an overlap between the potential data read by threads working on adjacent mesh cells. Thus, fetching the potential data into shared memory could have a positive impact on performance. However, our tests for the 2-point approximation, performed at the early stages of the program's development, did not indicate a significant improvement in the algorithm's running time, and this optimization path was not explored further.

A significant performance bottleneck in the GPU implementation is the transfer of data between the host and the device. These data transfers are necessary if one wishes to record the state of the simulation after each timestep. Suppose only the final state of the simulation is of interest. In that case, the PM method can execute entirely on the device, transferring the data to the host only once after the final iteration of the simulation. This is the scenario that we assumed to compare the CPU and GPU implementation. The results of the test for different particle numbers are shown in Figure 2.8. In the

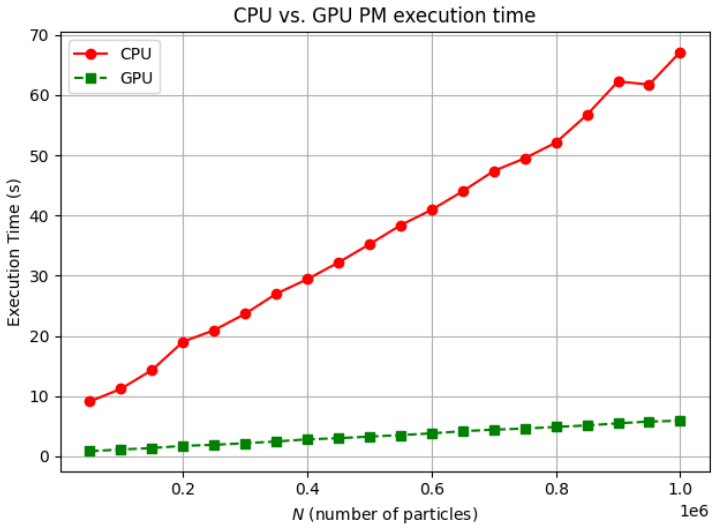


Figure 2.8: PM method: CPU and GPU implementation running time comparison (200 iterations; intermediate state not saved).

test, the computational domain was split into $64 \times 64 \times 32$ cells, and the simulation was run for 200 iterations. Both plots in the figure exemplify the linear complexity of the PM method, with the GPU implementation providing 1200% speedup over the CPU version.

Chapter 3

Particle-particle particle-mesh method

The P^{3}M algorithm is a hybrid method: Forces between distant particles are calculated using the PM method, whereas, for particles lying closely together, the PP method is used. The total force applied to particle i is

$$\mathbf{F}_i^{\mathrm{SR}} + \mathbf{F}_i = \sum_{j \neq i} (\mathbf{f}_{ij}^{\mathrm{tot}} - \mathbf{R}_{ij}) + \mathbf{F}_i, \tag{3.1}$$

where $\mathbf{F}_i \approx \sum_{j \neq i} \mathbf{R}_{ij}$ is the force computed using the PM method and $\mathbf{R}_{ij} = \mathbf{R}(\mathbf{x}_i - \mathbf{x}_j)$ is a prescribed *reference force*. The reference force is defined as the force between two particle-clouds, i.e. each particle is represented by a sphere with diameter a and a given density profile. The two examples of reference forces described in [8] are

$$R(r) = Gm_1m_2 \times \begin{cases} \frac{1}{35a^2}(224\xi - 224\xi^3 + 70\xi^4 + 48\xi^5 - 21\xi^6), & 0 \leq \xi \leq 1 \\ \frac{1}{35a^2}(12/\xi^2 - 224 + 896\xi - 840\xi^2 + 224\xi^3 + 70\xi^4 - 48\xi^5 + 7\xi^6), & 1 < \xi \leq 2 \\ \frac{1}{r^2}, & \xi > 2 \end{cases}$$

where $\xi = 2r/a$ for a sphere with uniformly decreasing density (shape S_2) and

$$R(r) = Gm_1m_2 \times \begin{cases} \frac{1}{a^2}(8r/a - 9r^2/a^2 + 2r^4/a^4), & r < a \\ \frac{1}{r^2}, \end{cases} \tag{3.2}$$

for a solid sphere (shape S_1). The reference force vector lies along the line joining the two bodies.

3.1 Optimal Green's function

As it is apparent from Equation 3.1, the method's validity depends on how well the mesh force approximates the reference force. The average deviation between the two forces can be minimized by a suitable choice of Green's function given (in k space) by

$$\hat{\mathcal{G}}(\mathbf{k}) = \frac{\hat{\mathbf{D}}(\mathbf{k}) \cdot \sum_{\mathbf{n}} \hat{U}^2(\mathbf{k_n}) \hat{\mathbf{R}}(\mathbf{k_n})}{|\hat{\mathbf{D}}(\mathbf{k})|^2 \left[\sum_{\mathbf{n}} \hat{U}^2(\mathbf{k_n})\right]^2}. \tag{3.3}$$

The derivation is mathematically involved and is detailed in [8]. We now proceed to examine the terms appearing in Equation 3.3.

3.1.1 Finite difference operator

The Fourier transform $\hat{\mathbf{D}}$ of the two-point finite difference operator defined in Equation 2.10 has the components

$$\begin{aligned}\hat{D}_j(\mathbf{k}) &= \int D_j(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}} d\mathbf{x} \\ &= \frac{1}{2H}\left(\int \delta(\mathbf{x}+H\mathbf{e}_j)e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x} - \int \delta(\mathbf{x}-H\mathbf{e}_j)e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}\right) \\ &= \frac{1}{2H}(e^{ik_jH} - e^{-ik_jH}) = \frac{i\sin(k_jH)}{H}.\end{aligned}$$

Similarly, for the four-point finite difference (Equation 2.11) we have

$$\hat{D}_j = \alpha\frac{i\sin k_jH}{H} + (1-\alpha)\frac{i\sin 2k_jH}{2H},$$

where $j = 1, 2, 3$.

3.1.2 Assignment function

The quantity $\hat{U}$ is defined as $\hat{W}/V$. For the mass assignment scheme hierarchy described in section 2.1 we have

$$\hat{U}(\mathbf{k}) = \left(\prod_{i=1}^{3}\frac{\sin(k_iH/2)}{k_iH/2}\right)^p,$$

where $p = 1, 2, 3, \ldots$ with $p = 1$ corresponding to NGP assignment, etc. In particular, for the TSC assignment scheme, the *alias sum*[1]

$$\sum_{\mathbf{n}}\hat{U}^2(\mathbf{k_n}) \equiv \sum_{\mathbf{n}}\hat{U}^2\left(\mathbf{k}+\mathbf{n}\frac{2\pi}{H}\right)$$

can be rewritten as

$$\begin{aligned}\sum_{\mathbf{n}}\hat{U}^2\left(\mathbf{k}+\mathbf{n}\frac{2\pi}{H}\right) &= \sum_{\mathbf{n}}\prod_{i=1}^{3}\left[\frac{\sin(k_iH/2+n_i\pi)}{k_iH/2+n_i\pi}\right]^6 \\ &= \prod_{i=1}^{3}\sum_{n_i}\left[\frac{\sin(k_iH/2+n_i\pi)}{k_iH/2+n_i\pi}\right]^6 \\ &= \prod_{i=1}^{3}\left[\sin^6\left(\frac{k_iH}{2}\right)\sum_{n_i}\frac{1}{(k_iH/2+n_i\pi)^6}\right]\end{aligned}$$

Using the partial fractions expansion of the cotangent function [2],

$$\frac{(-1)^s}{s!}\frac{d^s}{dx^s}\cot x = \sum_{n=-\infty}^{\infty}\frac{1}{(x-n\pi)^{s+1}},$$

[1] To get the alias sums compatible with the DFT definition given in Equation 2.4, one has to compute

$$\sum_{\mathbf{n}}\tilde{U}^2(\mathbf{k_n}) \equiv \sum_{\mathbf{n}}\tilde{U}^2(\mathbf{k}+\mathbf{n}N) = \frac{1}{H}\sum_{\mathbf{n}}\hat{U}^2\left(\mathbf{k}\frac{2\pi}{NH}+\mathbf{n}\frac{2\pi}{H}\right)$$

instead.

we can simplify the sum over n_i to

$$\frac{-1}{5!}\frac{d^5}{dx^5}\cot\left(\frac{k_iH}{2}\right) = 1 - \sin^2\frac{k_iH}{2} + \frac{2}{15}\sin^4\frac{k_iH}{2}.$$

Hence,

$$\sum_{\mathbf{n}} \hat{U}^2_{\mathrm{TSC}}(\mathbf{k_n}) = \prod_{i=1}^{3}\left(1 - \sin^2\frac{k_iH}{2} + \frac{2}{15}\sin^4\frac{k_iH}{2}\right).$$

Using the same approach, we can obtain similar results for the CIC and NGP schemes, namely

$$\sum_{\mathbf{n}} \hat{U}^2_{\mathrm{CIC}} = \frac{1}{27}\prod_{i=1}^{3}\left(1 + 2\cos^2\frac{k_iH}{2}\right) \quad \text{and} \quad \sum_{\mathbf{n}} \hat{U}^2_{\mathrm{NGP}} = 1.$$

3.1.3 Reference force

The quantity $\hat{\mathbf{R}}$, the transformed reference force, is related to the shape S of the particle-cloud by

$$\hat{\mathbf{R}}(\mathbf{k}) = \frac{i\mathbf{k}\hat{S}^2(k)}{k^2}, \tag{3.4}$$

where $k = |\mathbf{k}|$. This can be shown by recalling that $\mathbf{R}(\mathbf{x}_i - \mathbf{x}_j)$ is the force applied to cloud i due to cloud j. Hence, $\mathbf{R}(\mathbf{x})$ is the force on the cloud centered at $\mathbf{x}$ due to a cloud at the origin. The force acting on mass element $d\mathbf{x}'S(\mathbf{x}' - \mathbf{x})$ of the cloud centered at $\mathbf{x}$ is therefore

$$-Gd\mathbf{x}'S(\mathbf{x}' - \mathbf{x})\int d\mathbf{x}''S(\mathbf{x}'')\frac{\mathbf{x}' - \mathbf{x}''}{|\mathbf{x}' - \mathbf{x}''|^3}$$

and the total force is

$$\mathbf{R}(\mathbf{x}) = -G\int d\mathbf{x}'S(\mathbf{x}' - \mathbf{x})\int d\mathbf{x}''S(\mathbf{x}'')\frac{\mathbf{x}' - \mathbf{x}''}{|\mathbf{x}' - \mathbf{x}''|^3}. \tag{3.5}$$

The expression on the right-hand side of Equation 3.5 is a double convolution, $\mathbf{R} = -S * (S * \mathbf{g})$, where $\mathbf{g}(\mathbf{x}) = \mathbf{x}/|\mathbf{x}|^3$. The x-coordinate of $\mathbf{g}$ is $x/|\mathbf{x}|^3$ which coincides with $\partial h/\partial x$ for $h(\mathbf{x}) = -1/|\mathbf{x}|$. The Fourier transform of h is given in [6] (p. 363):

$$\hat{h}(\mathbf{k}) = \frac{4\pi}{|\mathbf{k}|^2}.$$

The formula for the transform of a derivative yields

$$\hat{g}_x(\mathbf{k}) = \frac{4\pi ik_x}{|\mathbf{k}|^2}.$$

Applying the convolution theorem twice and factoring the constant $(-4\pi G)$ out of the Green's function in Equation 3.3 leaves us with the desired Equation 3.4.

Since the shapes S are spherically symmetric, the calculation of $\hat{S}$ (appearing in Equation 3.4) can be simplified. The Fourier transform of S is

$$\hat{S}(\mathbf{k}) = \int S(\mathbf{x})e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}.$$

Observe that $\mathbf{k}\cdot\mathbf{x} = kr\cos\theta$, where θ is the angle between $\mathbf{k}$ and $\mathbf{x}$, and $r = |\mathbf{x}|$. Since S is invariant under rotations, θ can be chosen to be the angle between $\mathbf{x}$ and the z-axis. Thus, the integral, rewritten in spherical coordinates, becomes

$$\hat{S}(k) = \int_0^{2\pi} d\phi\int_0^{\pi} d\theta\int_0^{\infty} drS(r)e^{-ikr\cos\theta}r^2\sin\theta = 2\pi\int_0^{\infty} r^2dr\int_0^{\pi} d\theta e^{-ikr\cos\theta}\sin\theta.$$

The θ-integral upon the substitution $-kr\cos\theta \to u$ becomes

$$\frac{1}{kr}\int_{-kr}^{kr} e^{iu}du = \frac{1}{kr}\int_{-kr}^{kr}(\cos u + i\sin u)du = \frac{2\sin kr}{kr}$$

and hence

$$\hat{S}(k) = 4\pi\int_0^\infty r^2 S(r)\frac{\sin kr}{kr}dr.$$

This integral, evaluated for the S_1 and S_2 shapes respectively, gives

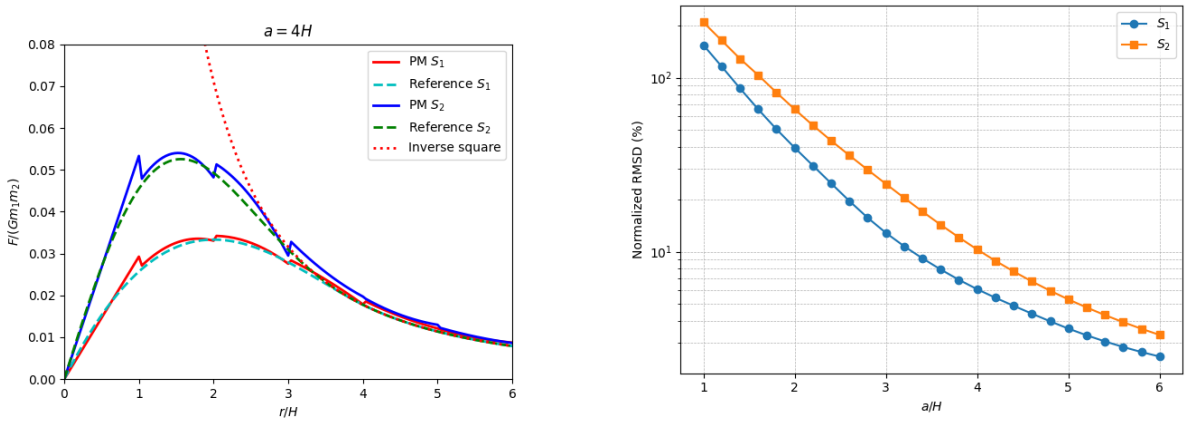
$$\hat{S}_1(k) = \frac{3}{(ka/2)^3}\left(\sin\frac{ka}{2} - \frac{ka}{2}\cos\frac{ka}{2}\right)$$

and

$$\hat{S}_2(k) = \frac{12}{(ka/2)^4}\left(2 - 2\cos\frac{ka}{2} - \frac{ka}{2}\sin\frac{ka}{2}\right).$$

Finally, we note that the infinite sum in the numerator of Equation 3.3 does not have a closed form, but this does not pose a problem since the summand decays rapidly with $\mathbf{n}$ moving further away from zero.

The result of applying the optimal Green's function in Equation 2.5 is illustrated in Figure 3.1a. As can be seen, the PM force closely follows the reference force, especially for distances $r > a$, where the reference force nearly matches the inverse-square law. It is worth noting that for r slightly smaller than a, the reference force and its mesh-based PM approximation still provide a good fit to the inverse-square behavior. This observation allows the cutoff radius r_e, which defines the boundary for direct summation, to be chosen smaller than a (e.g., $r_e = 0.7a$), improving performance without sacrificing accuracy.



(a) Approximate vs. reference force.

(b) Mean-normalized RMSD error

Figure 3.1: Comparison of PM approximation using S_1 and S_2 shape functions. The mesh approximation to the reference force was computed using the PM method with a TSC assignment scheme, two-point finite difference, and Green's function optimal for each shape.

3.1.4 Error analysis

The P^{3}M method offers a high degree of flexibility in terms of specifying the "building blocks" of the Green's function in Equation 3.3. For example, we can be interested in learning how the choice of the

assignment scheme influences the shape of the PM-approximated reference force. The result of a test that addresses this issue is shown in Figure 3.2. The figure shows that the general characteristic of the

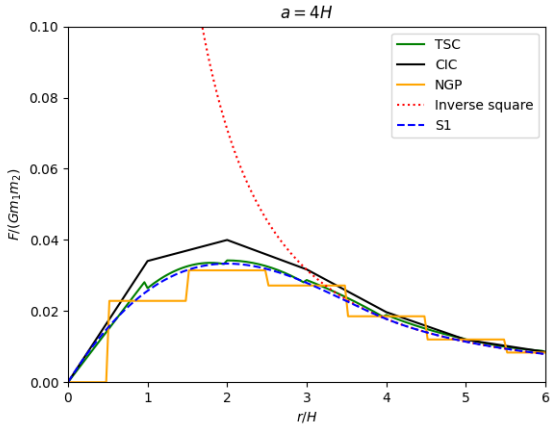


Figure 3.2: Approximate vs. reference force.

field strength approximated by the PM method is similar to what we have already seen in Figure 2.4b. Namely, the NGP scheme yields an unnatural, jagged shape. The shape produced using the CIC shape matches the S_1 reference force more closely, although it overestimates the field strength values close to the source. The most accurate approximation is obtained using the TSC scheme (this is the same shape we have already seen in Figure 3.1a).

To measure the quality of PM-approximation of a given reference force, we used mean-normalized RMSD error, i.e., we calculated

$$\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(F^{\mathrm{PM}}(r_i) - R(r_i))^2}}{\frac{1}{n}\sum_{i=1}^{n} R(r_i)}$$

in a system with a single unit mass located on the x-axis. The forces $F^{\mathrm{PM}}(r_i)$ were measured along the x-axis in equal-length intervals, and the test was run for increasing values of particle diameter a. The result of the test is shown in Figure 3.1b. Clearly, the approximation error becomes smaller for increasing values of a. Since r_e is proportional to a, this implies that there is a trade-off between accuracy and the execution time. Increasing a means that r_e should increase as well, which forces more particles into short-range correction regions. If the system under consideration has uniform density, we note that for any given particle, the short-range correction is concerned with $\sim r_e^3$ particles in its vicinity, which translates into $\sim r_e^3$ operations. The relation between cutoff radius size r_e and execution time is shown in Figure 3.3. As we can see, the execution time very strongly depends on r_e, and this explains why choosing r_e slightly, even slightly less than a, can lead to significant improvements in performance. Another conclusion that can be drawn from Figure 3.1b is that the S_2-based approximation exhibits higher mean-normalized RMSD error compared to S_1. However, as we remarked previously, it allows for choosing $r_e \lesssim a$ without significant loss of accuracy. Therefore, the choice between S_1 and S_2 shapes involves a trade-off between accuracy near the source and overall error, making the optimal choice context-dependent.

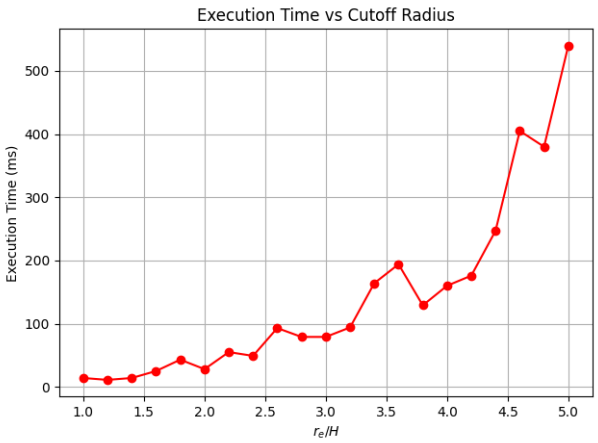


Figure 3.3: Execution time for different setting of cutoff radius r_e in the P^{3}M method (single step, $N = 10{,}000$). The times shown in the graph do not include tabulating the Green's function and the PM step.

3.2 Identifying close pairs of particles

In the P^{3}M method, in addition to the mesh used in the PM algorithm (the "potential mesh"), a second mesh (the *chaining mesh*) is used. The chaining mesh is sparser than the potential mesh. Its sole purpose is to partition the space into cells so that particles "close" to the ones in a given cell can be found efficiently. In this context, two particles are considered to lie close to one another if their separation is less than the cutoff radius.

The number of chaining mesh cells in a single dimension is given by $M_i = \lfloor L_i/r_e \rfloor$, where L_i is the side length of the computational box ($i = 1, 2, 3$). This implies that the side length of a chaining mesh cell is $\mathrm{HC}_i = L_i/M_i \geq r_e$. Thus, for every particle i in a given cell $\mathbf{p}$, it is sufficient to search through the immediate neighborhood of $\mathbf{p}$ to find all the particles within the cutoff radius from i.

In our program, the chaining mesh is implemented as a *head-of-chain* (HOC) array, depicted in Figure 3.4. The basic version of the HOC array is very cheap to build. Given a particle at point $\mathbf{x} =$

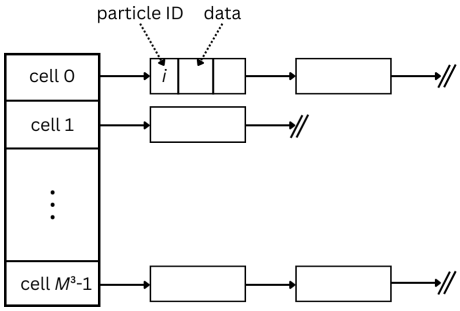


Figure 3.4: Head-of-chain data structure used for mapping particles to their parent cells in the chaining mesh. Here $M_1 = M_2 = M_3 = M$.

(x, y, z), we can determine in which chaining mesh cell $\mathbf{c}$ it lies by simply performing integer division on each of its coordinates, i.e. $\mathbf{c} = (x/\mathrm{HC}_1, y/\mathrm{HC}_2, z/\mathrm{HC}_3)$. Then, to get a HOC table index i corresponding

to $\mathbf{c}$, standard index flattening is used. Subsequently, a new linked-list node is allocated and inserted at the beginning of the list HOC[i]. The whole process is linear in the number of particles, and the array can be constructed anew at each time step without degrading performance. In our implementation, additional savings are made by preallocating a memory pool large enough to store N nodes of the linked lists and reusing it for the HOC array initialization.

A potential optimization discussed in [8] involves sorting each linked list by a selected particle coordinate, such as the y-coordinate. This ordering enables early termination of the direct summation loop when the condition $y_i - y_j > r_e$ is met, effectively reducing unnecessary pairwise computations as particle i "sweeps through" a cell containing particles j. Our experiments indicate that constructing the linked lists in sorted order is substantially more expensive than the unsorted variant. This is to be expected; in the worst-case scenario where all particles are in a single chaining-mesh cell and are inserted into the list in decreasing order of y coordinates, the complexity is $O(N^2)$. For a system of 50,000 particles, incorporating y-sorting increased the HOC (head-of-chain) construction time from approximately 180 milliseconds to over 13,000 milliseconds, a 70-fold slowdown. Despite this added cost, we observed no significant performance improvement in the short-range correction phase of the computation.

3.3 Short-range correction

The short-range correction, which takes place immediately after the mesh forces are found using the PM method, is at the heart of the P^{3}M algorithm. Since it scales with a square of the number of particles in each neighborhood, further optimizations are highly desirable.

By Newton's 3rd law, $\mathbf{f}_{ji}^{\mathrm{SR}} = -\mathbf{f}_{ij}^{\mathrm{SR}}$, which allows us to do the calculation of the short-range inter-particle force for any pair (i, j) of particles only once, leading to the reduction of the total running time by half. Informally, the particle i updates its total short-range force $\mathbf{F}_i^{\mathrm{SR}}$ as well as the total short-range force $\mathbf{F}_j^{\mathrm{SR}}$ of its neighbor j. To avoid double-counting, the particle i residing in cell $\mathbf{q}$ has to look for its neighbors in a subset $\mathcal{N}$ of the immediate neighborhood of $\mathbf{q}$. More specifically, define

$$\mathcal{N}(\mathbf{q} = (q_1, q_2, q_3)) = \{(q_1 + t, q_2 - 1, q_3 + s), (q_1 + s, q_2, q_3 - 1), (q_1 - 1, q_2, q_3) \mid s, t = -1, 0, 1\}. \quad (3.6)$$

Thus $|\mathcal{N}| = 13$, which is half of the size of the immediate neighborhood. The set $\mathcal{N}$ given in Equation 3.6 is not easy to illustrate. For the later discussion, it is beneficial to have a clear picture in mind; therefore, we depict its two-dimensional analog in Figure 3.5.

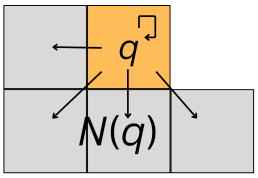


Figure 3.5: Set $\mathcal{N}$ in two dimensions.

The short-range correction part of the P^{3}M method is shown in Algorithm 3. For each chaining mesh

Algorithm 3 Short-range correction

```
1: for each chaining cell q do
2:     for each q_n ∈ N(q) ∪ {q} do
3:         for each i ∈ HOC(q) do
4:             for each j ∈ HOC(q_n) do
5:                 if |y_i − y_j| > r_e then
6:                     break
7:                 UpdateShortRange(i, j, q, q_n)
```

cell $\mathbf{q}$, we compute the pair-wise interactions between the particles in $\mathbf{q}$ and the particles in the chaining mesh cells $\mathbf{q}_n$ from the reduced neighborhood $\mathcal{N}(\mathbf{q})$ *plus* $\mathbf{q}$. The inclusion of $\mathbf{q}$ allows us to take into account forces between particles within $\mathbf{q}$.

The UpdateShortRange procedure is defined in Algorithm 4. In line 2, we exclude self-forces.

Algorithm 4 Updating short-range forces

```
1: procedure UpdateShortRange(i, j, q, q_n)
2:     if i = j then return
3:     r_ij ← r_i − r_j
4:     if |r_ij|^2 > r_e^2 then return
5:     r_ij ← |r_ij|
6:     r̂_ij ← r_ij / r_ij
7:     R_ij ← −m_i m_j R(r_ij) r̂_ij
8:     f^tot ← −G m_i m_j / r_ij^2 r̂_ij
9:     f_ij^SR ← f^tot − R_ij
10:    f_ji^SR ← −f_ij^SR
11:    F_i^SR ← F_i^SR + f_ij^SR
12:    if q_n ≠ q then                ▷ Avoid double-counting in the parent cell
13:        F_j^SR ← F_j^SR + f_ji^SR
```

The check in line 4 assures that the correction happens only for particles with separation less than the cutoff radius r_e. The pair-wise short-range force is calculated in lines 5–10, with the calculation in line 10 exploiting Newton's third law, as described previously. The accumulation of total short-range force for a given particle takes place in line 11. The short-range force is also added to the total short-range force of the other particle in the pair (particle j) but only if $\mathbf{q}_n \neq \mathbf{q}$. This stipulation is crucial, as otherwise, we would be double counting the forces between particles within $\mathbf{q}$.

As suggested in [8], the computational burden of operations in lines 5–8 in Algorithm 4 can be greatly reduced by storing the values of $f^{\mathrm{SR}}(r)/r = (f^{\mathrm{tot}}(r) - R(r))/r$ in a lookup table T at uniform intervals Δ^2 of $[0, r_e^2]$ and interpolating them later. The schematic depiction of the interpolation is shown in Figure 3.6. If we define $\xi = r^2/\Delta^2$ and $t = \lfloor \xi \rfloor$, then

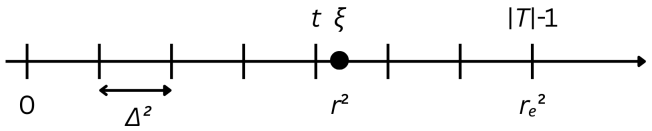


Figure 3.6: Interpolation of short-range force values.

$$\frac{f^{\mathrm{SR}}(r)}{r} \approx T[t]\,(1-(\xi-t)) + T[t+1](\xi-t) = T[t] + (\xi-t)(T[t+1]-T[t]).$$

The value $\mathbf{f}_{ij}^{\mathrm{SR}}$ can then be obtained by multiplying the interpolated quantity $f^{\mathrm{SR}}(r_{ij})/r_{ij}$ by $Gm_im_j\mathbf{r}_{ij}$, eliminating the use of the square root operations and reducing the total number of floating-point operations to just four.

3.3.1 Parallelization

In our implementation, the procedure outlined inAlgorithm 3 is parallelized by splitting the work done in the outmost loop between some number of threads. In doing so, extra care has to be taken to avoid data races. A thread t that is currently processing cell $\mathbf{p}$ and its neighbors (we say that t is *assigned* to $\mathbf{p}$) may "clash" with a different thread assigned to a nearby cell $\mathbf{q}$ (because possibly $\mathbf{p} \in \mathcal{N}(\mathbf{q})$). However, by the construction of the set $\mathcal{N}$, it is possible to split the short-range force into 14 parts, each of which is accessed by only one thread. For example, consider a particle i in cell $\mathbf{p} = (p_1, p_2, p_3)$ (in other words, $\mathbf{p}$ is the parent cell of i). If thread t is currently assigned to this cell, t will update the part of $\mathbf{F}_i^{\mathrm{SR}}$ corresponding to updates of i coming from within the same cell as the parent cell of i. Possibly at the same time, thread t' assigned to cell $\mathbf{q} = (p_1 + 1, p_2, p_3)$ will update a different part of $\mathbf{F}_i^{\mathrm{SR}}$, i.e., the part corresponding to updates of i coming from the cell "to the right" of the parent cell of i. Since only one thread is responsible for updates to particle i coming "from the right" (or any other "direction"), no data races can occur. This approach presents two major drawbacks. First, it significantly increases memory usage, requiring storage for an additional $13N$ three-dimensional vectors. Second, it offers no guarantee of uniform workload distribution across threads. This imbalance, combined with the substantial variation in operations performed by individual threads, leads to severe thread divergence, rendering the parallelization scheme unsuitable for GPU execution, making it only applicable to the CPU implementation. Despite these shortcomings, tests performed on our machine (utilizing the maximum number of concurrent threads supported, equal to 12) yielded more than four-fold speedup compared to the single-threaded implementation in a typical simulation. The results of a performance test for a typical simulation are shown in Figure 3.7. As illustrated in the figure, increasing the number of threads initially improves performance, but beyond a certain point, the gains level off. This is because the system reaches the limit of its physical cores (six in our case) after which additional threads do not contribute meaningfully. Further increasing the thread count introduces overhead from context switching, which ultimately becomes the dominant performance bottleneck.

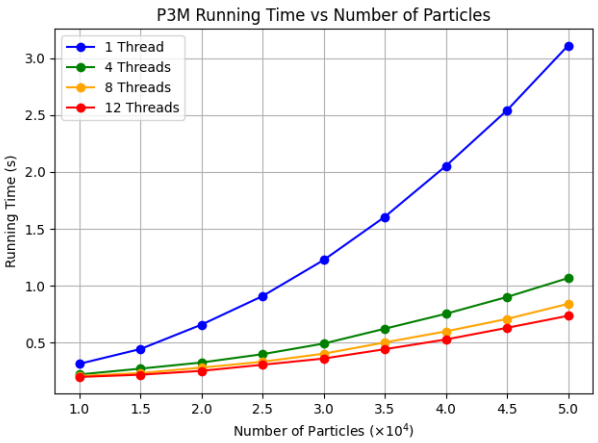


Figure 3.7: Running time per iteration of the P^{3}M method using different numbers of threads.

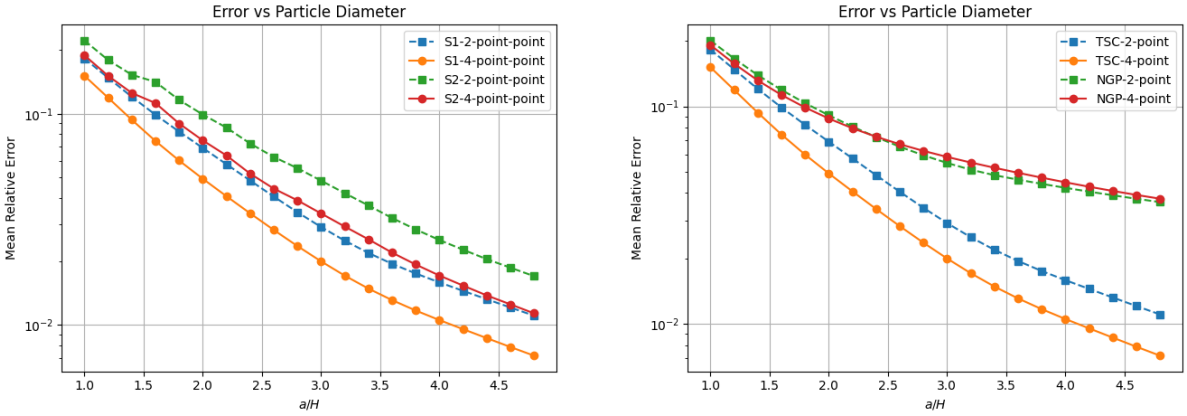
3.4 Global force approximation error

The situation portrayed in Figure 3.2 gives insight into a field due to a single particle. In a typical situation, we are interested in simulating many thousands of particles, and thus, the averaged approximation error becomes of interest. Just as in the case of the PM method, the global error was measured in terms of mean relative error given by Equation 2.12. This time, the error was plotted as a function of the particle diameter for both TSC and NGP schemes in two variants, using second-order and fourth-order Laplacian approximations. The result of this test is shown in Figure 3.8b. The figure illustrates that the error decays rapidly with increasing particle diameter when the TSC scheme is used. It is worth noting that the fourth-order accurate finite difference yields significant improvement in accuracy for the TSC scheme-based P3M method, allowing the error to drop below 1% mark for $a \approx 4H$.

The "local picture" of the errors of force approximation in the P^{3}M method has already been outlined in Figure 3.1. Now we turn to investigate how the choice of the particle shape (S_1 or S_2) and the size of the particle diameter influence the global force approximation error. The error is defined the same as in the case of the PM method, i.e., we use Equation 2.12.

In our test, the grid covering the computational region had dimensions of $64 \times 64 \times 32$, and $N = 10{,}000$ particles were used. The result of the test is shown in Figure 3.8. The figure clearly illustrates that the error drops with increasing the size of the particle diameter a, which is an expected result, as the P3M method calculates more interparticle forces with increasing a through exact direct summation. Moreover, as we could see in Figure 3.1b, the PM approximation to the reference force gets better with increasing particle diameter. These two effects combined are responsible for the reduction of global error.

For both the S_1 and S_2 shapes, the error was minimized when the fourth-order finite difference was used for gradient calculation, as one could expect. The test indicated the superiority of the S_1 shape, with the difference between the two becoming more significant for bigger relative values of a.



(a) Mean error for different particle shapes.

(b) Mean error for different assignment schemes.

Figure 3.8: Comparison of PM approximation using different assignment and interpolation functions. In the test we set $N = 10{,}000$ and used $64 \times 64 \times 32$ grid.

Chapter 4

Barnes-Hut algorithm

The idea behind the Barnes-Hut algorithm differs substantially from previously described mesh-based methods. The algorithm deals with gravitational forces directly instead of deriving them from the mesh-defined potential, as was the case with the PM and P^3M methods. Significant reduction of time complexity, from quadratic to $O(N \log N)$, is achieved by approximating the potential due to "far enough" groups of particles by the initial terms of its multipole expansion [15]. The grouping of particles is hierarchical in nature and is thus best understood as a tree. The entire set of particles comprises the top-level group, represented by the root of the tree; the eight children of the root node are representative of groups of particles residing in each of the octants of the computational domain, etc. The process of subdividing the space into eight smaller volumes at each node continues recursively until there are only one or zero particles left in a given volume. Nodes that satisfy this condition are the leaves of the tree and are sometimes called the *external nodes*. The remaining nodes, each of which has eight children, are called *internal nodes*.

In the basic variant of the algorithm, the potential due to a group of particles is approximated using only the monopole term with respect to the center of mass of the group, i.e.

$$\phi_{\text{mon}}(r) = -\frac{GM}{r},$$

where M is the group's total mass. Since the potential is expanded about the center of mass, the dipole moment $\mathbf{p} = \sum_i m_i \mathbf{r}_i$ vanishes. Hence, the next possible improvement comes from including the quadrupole term

$$\phi_{\text{quad}}(r) = -\frac{G}{2r^5}\mathbf{r} \cdot (\mathbf{Q}\mathbf{r}),$$

where $\mathbf{Q}$ is the quadrupole moment tensor defined as

$$Q_{ij} = \sum_k (3r_{ki}r_{kj} - 3r_k^2\delta_{ij})m_k.$$

In theory, we could keep on adding more terms to improve the quality of the approximation. In our implementation, however, we restrict ourselves to the quadrupole term.

4.1 Building the tree

The data structure that fits the description given in the introduction is called an *octree*. An internal node of the octree stores the COM vector, the total mass of the group it represents, and the quadrupole tensor, whereas an external node stores a reference to the actual particles (or is empty if no particle was found in its associated volume). The recursive procedure of building the tree is shown in Algorithm 5. The quadrupole moment tensor for each node is calculated once the whole tree is already built. The

Algorithm 5 Insert a particle into the Barnes-Hut tree

```
1: function INSERT(n, p)
2:     if n is an internal node then
3:         Update n.COM and total mass n.M of n with p
4:         INSERT(child of n that should contain p, p)
5:     else if n is empty then
6:         Assign p to n
7:     else                                          ▷ Occupied external node
8:         Subdivide n into child nodes
9:         Move existing particle p′ in n into child that should contain p′
10:        Update center of mass and total mass of n with p and p′
11:        INSERT(child of n that should contain p, p)
```

recursive relation used in this calculation is given in [7] and reads

$$\mathbf{Q} = \sum_{\text{child } c} \mathbf{Q}_c + \sum_{\text{child } c} m_c(3\mathbf{R}_c \otimes \mathbf{R}_c - R_c^2\mathbf{I}),$$

where $\mathbf{R}_c = \mathbf{x}_c^{\text{COM}} - \mathbf{x}^{\text{COM}}$ is the displacement vector from the COM of child c to the COM of the parent, $\mathbf{I}$ is the identity matrix, and $\otimes$ denotes the outer product.

For reasons that will become apparent later, it can be beneficial to separate the COM calculation from the tree creation part. In such a case, the COM is calculated recursively using the relation

$$\mathbf{x}^{\text{COM}} = \frac{\sum_{\text{child } c} m_c \mathbf{x}_c^{\text{COM}}}{\sum_{\text{child } c} m_c} \tag{4.1}$$

after the tree has already been built (and obviously before the quadrupole moment calculation).

4.2 Acceleration calculation

In the Barnes-Hut algorithm, the net acceleration of a particle p is calculated by summing the contributions from single particles or groups of particles while traversing the tree. The decision of whether the acceleration can be approximated using the information stored in an internal node n depends on the relative distance from p to n.COM (the center of mass of the group represented by n). The distance is relative to the *width* H of the node, i.e. the side length of the cubical volume encompassed by the node. More concretely, the approximation takes place if $n.H/|n.\text{COM} - p.\mathbf{x}| < \theta$, where θ is the so-called *opening angle*. In the extreme case when θ is set to zero, no approximations take place, and the

algorithm reduces to the PP method. The procedure described above is illustrated in Algorithm 6. In

Algorithm 6 Compute gravitational force on a particle using Barnes-Hut approximation

```
1: function FindAcceleration(n, p, θ)
2:     if n is an external node then
3:         if n contains a particle q ≠ p then
4:             p.a ← p.a + GravitySoft(q.x, q.m, p.x)/p.mass
5:         return
6:     if n.H/|n.COM − p.x| < θ then
7:         p.a ← p.a + Gravity(n.COM, n.M, p.x)
8:         return
9:     for each child n_c of n do
10:        FindAcceleration(n_c, p, θ)
```

the implementation, the GravitySoft function calculates the gravitational force softened by ϵ, i.e. it returns the value given by Equation 2.13. The pairwise potential energy associated with this force is given by

$$\Phi_{ij}^{\text{soft}} = -\frac{Gm_i m_j}{\sqrt{r_{ij}^2 + \epsilon^2}}. \tag{4.2}$$

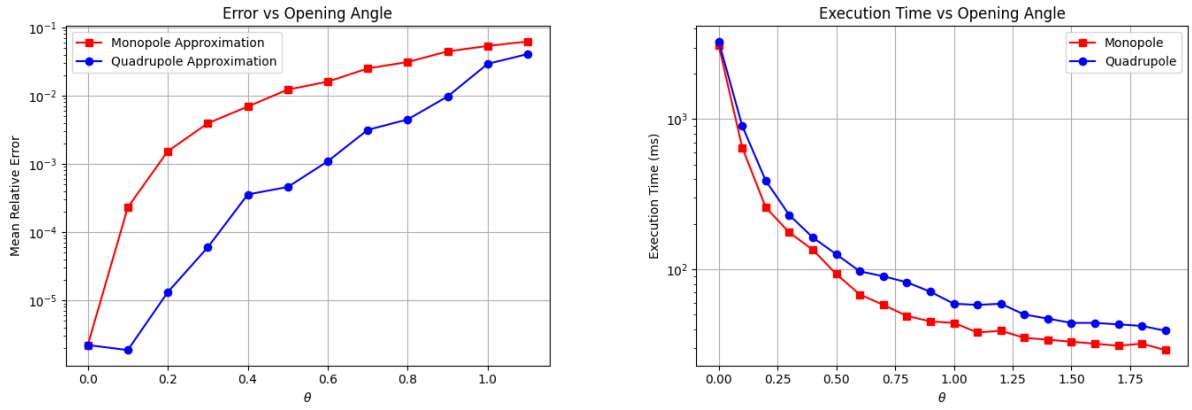
The Gravity function returns the approximation (up to the quadrupole term) of the acceleration due to a group of particles represented by a given node, i.e.

$$\mathbf{a} = -GM\frac{\mathbf{r}}{r^3} + \frac{G}{r^5}\mathbf{Q}\mathbf{r} - \frac{5G}{2}(\mathbf{r}\cdot(\mathbf{Q}\mathbf{r}))\frac{\mathbf{r}}{r^7}$$

(see [7]).

One possible way to quantify the quality of approximation for a given value of θ is to consider the relative error of calculated force. We set the same initial conditions of the system for both the PP direct summation method and the Barnes-Hut algorithm, compute the deviation of Barnes-Hut forces from PP forces acting on each particle, and take the average over all particles. In other words, the error calculated is given by the Equation 2.12. The dependence of the error on the opening angle θ and the corresponding execution time are shown in Figure 4.1. The figure includes plots for two cases: when only the monopole term is used in the Barnes-Hut approximation, and when both the monopole and quadrupole terms are included. As can be seen, the quadrupole-based algorithm exhibits significantly improved error scaling with increasing θ. Naturally, this raises the question of the additional computational cost incurred by the inclusion of the quadrupole term. Our tests showed that this impact is minimal. The results (for N = 10,000 particles and $0 \leq \theta \leq 2$) support this observation. Both tests described above were conducted on a uniform disk particle distribution.

We note that direct calculation of total potential energy is infeasible as $O(N^2)$ operations would be required. Instead, we use an approximation based on the values stored in the tree. The approximate value of the potential energy is accumulated for each particle using a procedure analogous to force calculation. Indeed, the only difference between the two is the replacement of gravitational force calculation in Algorithm 6 with potential energy calculation according to Equation 4.2.



(a) Force approximation error.

(b) Execution time per iteration.

Figure 4.1: Comparison of error and execution time in the Barnes-Hut algorithm using monopole and quadrupole approximations.

It is also noteworthy that the procedure outlined in Algorithm 6 is embarrassingly parallel. In our CPU implementation, the workload is split between an arbitrary number of threads on a particle-by-particle basis.

4.3 Accelerating tree construction

Unlike the force calculation part which is easily parallelized, the tree construction step is inherently sequential. An approach to parallel tree construction outlined in [17] involves splitting the set of particles into load-balanced spatial groups which are then used to construct separate trees (one per thread), with the final step being the merge of the created trees. The merge step is highly involved and beyond the intended scope of this thesis. For this reason, we propose an alternative approach to accelerating the tree construction which does not involve building the tree in parallel.

Our strategy, based on the ideas put forth in [17], stems from an observation that a potential bottleneck in Algorithm 5 is related to irregular accesses to the nodes of the tree. Inserting two particles separated by a large spatial distance leads to visiting nodes along two completely different paths in the tree which results in a huge number of cache misses. A solution to this problem is to sort the list of particles in such a way that subsequent particles on the list (typically) lie close to each other in the physical space. An ordering with this property can be generated by numbering the particles based on their position along a chosen *space-filling curve*.

A space-filling curve can be thought of as the limit of an infinite sequence of curves that "fill" the space without "holes" [19]. In the limit, the curve reaches every point of the space, but due to practical constraints, we can only deal with an approximation, i.e. some member of the limiting sequence. In our implementation, we are working with a *Z-order curve*, also called a *Morton space-filling curve*, so it will in the focus of our discussion. To make the visualization simpler, assume that the computational domain is two-dimensional and coarsely divided into 16 cells (particles within the same cells are ordered

arbitrarily). Then, the Z-order curve that covers all 16 cells is shown in Figure 4.2. The ordering of

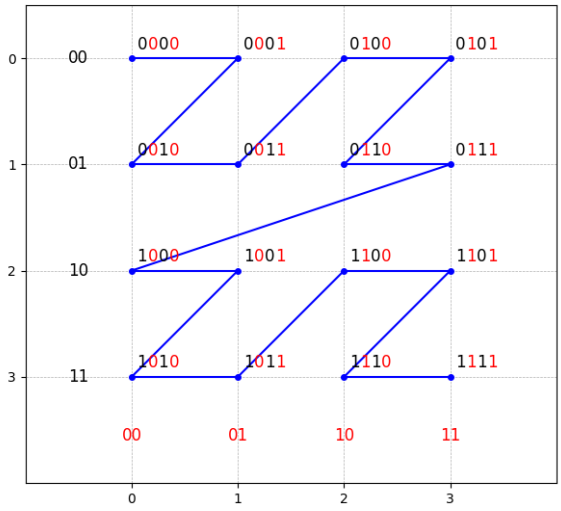


Figure 4.2: Z-order curve.

the points along the curve is obtained by assigning each cell a *Z-value*, whose binary representation is calculated by interleaving the bits of the x and y coordinates. This is illustrated in Figure 4.2 by coloring the x- and y-coordinate bits red and black respectively.

In a standard setting, the coordinates of a particle are given by real numbers and not integers, however. To establish a mapping between the two, we have to first decide on the intended resolution of the grid. In our implementation, we use the 32-bit integer type to represent the Z-values, which means that each coordinate should be represented by a 10-bit integer number ($3 \times 10 = 30 \leq 32$). Thus, the mapping from the real-valued coordinate of a particle to an integer one is given by

$$\left\lfloor \frac{(x - \text{low}.x)}{H} \times (2^{10} - 1) \right\rfloor \in [0, 2^{10}),$$

where low and H define the computational domain as

$$\text{domain} = [\text{low}.x, \text{low}.x + H] \times [\text{low}.y, \text{low}.y + H] \times [\text{low}.z, \text{low}.z + H].$$

In our proposed optimization, we make the following changes to the tree construction algorithm (Algorithm 5):

1. The insertion does no longer start at the root of the tree for each particle but at the last inserted to node,

2. The COM calculation is deferred until the tree construction is finished and carried out using Equation 4.1.

The second point is a direct consequence of the first one; if the insertion of a particle does not start at the root but at some other node n, deep in the tree, the nodes lying above n will not be updated so their COM and total mass values will be incorrect. The first point, however, requires more explanation.

Since the particles are z-ordered, subsequent inserts into the tree will result in traversing similar paths. Because of that, the point of insertion can be found more efficiently by starting the traversal from the last seen node and backtracking up the tree until a valid insertion point is found. The decision whether to stop backtracking at a given node n is made on the condition that the particle to be inserted is inside a cube represented by n. When the condition is met, the standard insertion procedure takes place starting at n.

The asymptotic time complexity of the modified tree construction algorithm is the same as for the standard one. Sorting the particles requires $O(N \log N)$ operations but inserting a particle into the tree starting at the last-seen node is an $O(1)$ operation. The last claim may raise objections since some number of backtrack steps are expected on each insert. To verify it, we measured the average number of backtracks per particle in typical simulations and found that it did not exceed 2.5 (this can be compared with the number of backtracks required when the particles are not z-ordered; in such a case, we found that in the tested scenarios, on average more than 8 backtracks were typically needed). Even though the modified algorithm still has the $O(N \log N)$ time complexity, it leads to more predictable memory access patterns which result in noticeably improved performance. The tree construction time as a function of N is shown in Figure 4.3. To generate the graphs shown there, the tree construction time was measured

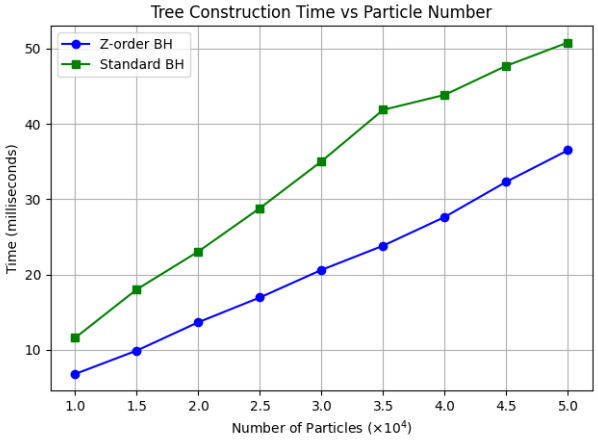


Figure 4.3: Tree construction time averaged over the number iterations of the simulation ($\theta = 1$).

in a spiral galaxy simulation. As can be seen in the figure, the improved algorithm allows for up to 40% speedup over the standard version. In our tests, we used the standard library procedure `std::sort` with the parallel execution policy, which offered a slight speedup over the sequential variant.

Chapter 5

Time integration

In the previous sections, we described various methods of calculating forces applied to particles in the simulation. Once these forces are found, the evolution of the system in time can be tracked by integrating Newton's 2nd law of motion,

$$\ddot{\mathbf{x}}_i = \frac{\mathbf{F}_i}{m_i}. \tag{5.1}$$

5.1 Euler's method

Possibly, the simplest numerical method that could be used is Euler's method described by the update rules

$$\begin{aligned} \mathbf{v}_i^{(k+1)} &= \mathbf{v}_i^{(k)} + \mathrm{DT}\frac{\mathbf{F}_i^{(k+1)}}{m_i}, \\ \mathbf{x}_i^{(k+1)} &= \mathbf{x}_i^{(k)} + \mathrm{DT}\mathbf{v}_i^{(k)}. \end{aligned} \tag{5.2}$$

The method defined in Equation 5.2 is not suitable for physical simulations, however. Its shortcomings are best illustrated by an example of an undamped pendulum of length l in a gravitational field of magnitude g. Although it is a straightforward system, it illustrates the numerical challenges faced in gravitational simulations over long timescales, particularly the issue of energy preservation.

The differential equation governs the motion of the pendulum

$$\ddot{\theta} = -\frac{g}{l}\sin\theta,$$

and its kinetic and potential energy are given by $\mathrm{KE} = (1/2)ml^2\dot{\theta}$ and $\mathrm{PE} = -mgl\cos\theta$ respectively. As shown in Figure 5.1, Euler's method fails to conserve total energy (PE + KE) and produces trajectories in phase space that are not closed, contrary to expectations for periodic systems. Additionally, the evolution of an area element in phase space violates Liouville's theorem, as described in [13], making the method unsuitable for long-term physical simulations (see Figure 5.2).

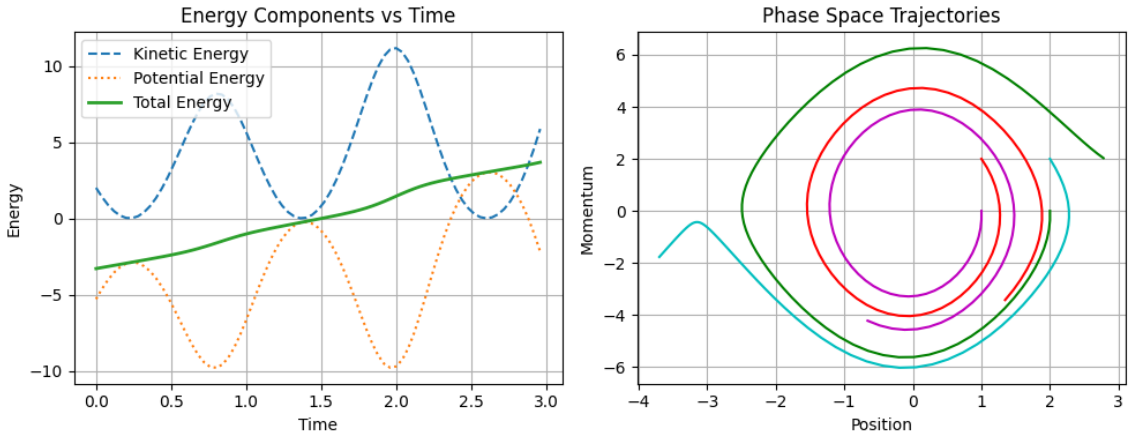


Figure 5.1: Behavior of Euler's method: lack of conservation of energy and phase space trajectories spiraling out.

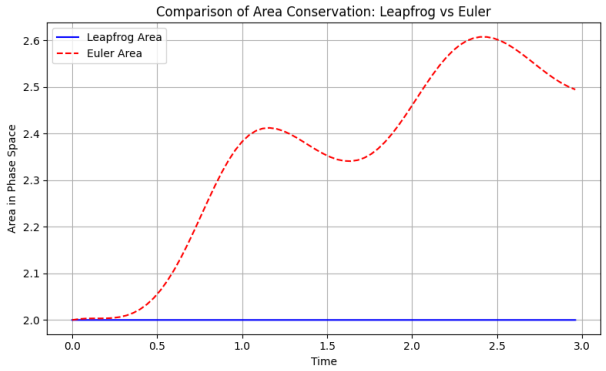


Figure 5.2: Area in phase space over time. Violation of Liouville's theorem by Euler's method.

5.2 Leapfrog algorithm

The leapfrog algorithm, given by the update rule [21]

$$\begin{aligned} \mathbf{v}_i^{(1/2)} &= \mathbf{v}_i^{(0)} + \frac{1}{2}\mathrm{DT}\frac{\mathbf{F}_i^{(0)}}{m_i^{(0)}}, \\ \mathbf{x}_i^{(k+1)} &= \mathbf{x}_i^{(k)} + \mathrm{DT}\mathbf{v}_i^{(k+1/2)}, \\ \mathbf{v}_i^{(k+3/2)} &= \mathbf{v}_i^{(k+1/2)} + \mathrm{DT}\frac{\mathbf{F}_i^{(k+1)}}{m_i}. \end{aligned} \tag{5.3}$$

is the preferred way of integrating Equation 5.1. When applied to the same pendulum system, it conserves energy much more faithfully. It preserves the area in phase space, consistent with Liouville's theorem (see Figure 5.3 and Figure 5.2). Given its simplicity and excellent long-term energy behavior, we adopt

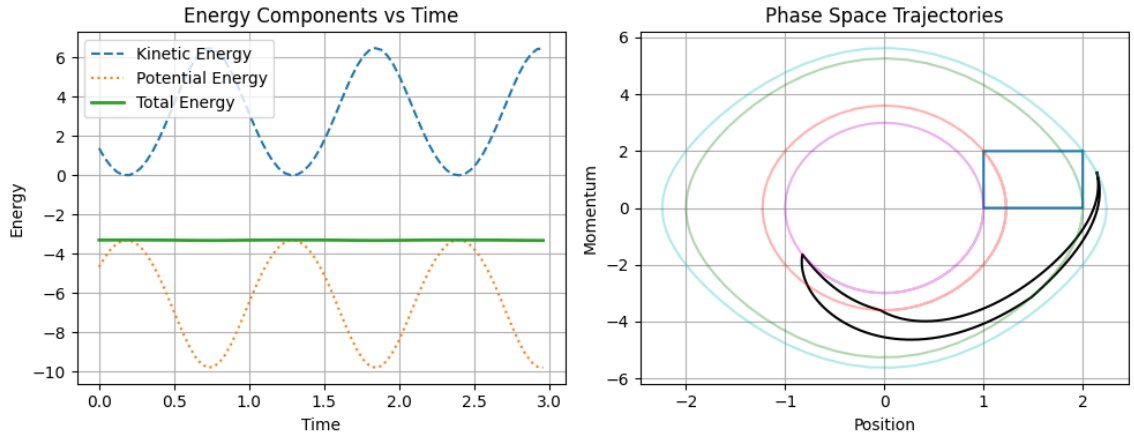


Figure 5.3: Behavior of the leapfrog algorithm: conservation of energy and phase space trajectories forming closed loops. The evolution of an area element in phase space is shown on the right-hand side: blue rectangle – initial conditions for many copies of the system; black distorted quadrilateral – their state by the end of the simulation.

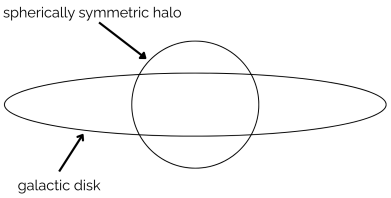
the leapfrog algorithm to integrate Newton's equations into our program.

Chapter 6

Test Models

6.1 Galaxy model

The model of a galaxy used as a test bed for the implementation is a simple one. The galaxy is assumed to comprise only two parts: a thin disk and a spherically symmetric halo. The disk comprises a large number of particles, each representing some number of stars. The halo is simulated as a fixed external gravitational field. The schematic illustration of the model is shown in Figure 6.1a. Figure 6.1b shows



(a) Spiral galaxy model (thin disk and spherical halo).

(b) Real-world spiral galaxy (NGC 3147).

Credit for Figure 6.1b: NASA, ESA, and the Hubble Heritage (STScI/AURA)-ESA/Hubble Collaboration; Acknowledgment: R. Chandar (University of Toledo) and J. Miller (University of Michigan).

Figure 6.1: Comparison between a modeled and a real-world spiral galaxy.

an example of a real-world spiral galaxy whose general structure we aim to reproduce.

6.1.1 Disk

The disk particles are sampled from a radial distribution

$$p(r) = \frac{3}{\pi R_D^2}\left(1 - \frac{r}{R_D}\right),\ z = 0,$$

where R_D is the radius of the disk and $r \leq R_D$. The cumulative distribution function is therefore

$$F(r,\phi) = \int_0^{\phi}\int_0^{r} p(r')r'dr'd\phi' = \frac{\phi}{2\pi R_D^3}(3R_D r^2 - 2r^3)$$

and the marginal CDFs are

$$F_R(r) = F(r, 2\pi) = \frac{1}{R_D^3}(3R_D r^2 - 2r^3) \quad \text{and} \quad F_\Phi(\phi) = F(R_D, \phi) = \frac{\phi}{2\pi}.$$

Now we use inverse transform sampling to generate initial positions (r, ϕ) for the particles, i.e. $\phi = 2\pi u$ and r is given implicitly by $h(r) \equiv 2r^3 - 3R_D r^2 + uR_D^3 = 0$ with $u \sim U(0,1)$. A straightforward calculation shows that $dh/dr < 0$ for $0 < r < R_D$ and $h(0)h(R_D) < 0$, implying that h has exactly one zero between 0 and R_D (which can be found for example using Newton's method).

Strength of the gravitational field $\mathbf{g}_D$ due to the disk at point $\mathbf{x}_0$ lying in the disk is

$$\mathbf{g}_D = G\int_0^{2\pi}\int_0^{R_D} \sigma(r)\frac{\mathbf{x} - \mathbf{x}_0}{|\mathbf{x} - \mathbf{x}_0|^3} r dr d\phi,$$

where $\sigma(r) = \sigma_0(1 - r/R_D)$ describes the density profile of the disk for $r \leq R_D$. If M_D is the total mass of the disk, then $\sigma_0 = 3M_D/(\pi R_D^2)$. By symmetry, the point $\mathbf{x}_0$ may be chosen to lie on the x-axis, i.e. $\mathbf{x}_0 = (-x_0, 0)$, so that $\mathbf{x} - \mathbf{x}_0 = (x_0 + r\cos\phi, r\sin\phi)$. Letting $\bar{r} = r/R_D$ and $\bar{x}_0 = x_0/R_D$, the integral becomes

$$\mathbf{g}_D = G\sigma_0\int_0^{2\pi}\int_0^{1}(1-\bar{r})\frac{(\bar{x}_0 + \bar{r}\cos\phi, \bar{r}\sin\phi)}{|(\bar{x}_0 + \bar{r}\cos\phi, \bar{r}\sin\phi)|^3}\bar{r}d\bar{r}d\phi.$$

By symmetry $g_{D,y} = 0$ and thus the radial component of the field $\mathbf{g}_D$ at distance $R\bar{x}_0$ from the center is

$$g_{D,r} = -|\mathbf{g}_D| = -G\sigma_0\int_0^{2\pi}\int_0^{1}(1-\bar{r})\frac{\bar{x}_0 + \bar{r}\cos\phi}{(\bar{x}_0^2 + \bar{r}^2 + 2\bar{x}_0\bar{r}\cos\phi)^{3/2}}\bar{r}d\bar{r}d\phi. \tag{6.1}$$

If the disk had constant density, $\mathbf{g}_D$ could be expressed in terms of elliptic integrals [18]. However, to the best of the author's knowledge, the integral in Equation 6.1 cannot be further simplified. For this reason, a crude approximation with a quadratic function is used: $g_{D,r} \approx a(r-h)^2 + k$, where the values $k = 2.5$ and $h = 0.66$ (the maximum of $g_{D,r}$ and the argument thereof) were estimated based on the graph of $g_{D,r}$ (see Figure 6.2). The value of $a = -k/h^2$ can be found by setting $g_{D,r}(0) = 0$ in the approximate formula.

6.1.2 Halo

The density profile of the halo is analogous to the one used for the disk, save for the fact it is 3-dimensional, i.e.

$$\rho(r) = \begin{cases} \rho_0\left(1 - \frac{r}{R_H}\right), & r \leq R_H \\ 0, & \text{otherwise,} \end{cases}$$

where R_H is the radius of the halo. If we let M_H be the mass of the halo, then $\rho_0 = 3M_H/(\pi R_H^3)$. Application of Gauss's law shows that we have

$$g_{H,r} = -GM_H \times \begin{cases} \frac{r}{R_H^3}\left(4 - \frac{3r}{R_H}\right), & r \leq R_H \\ \frac{1}{r^2}, & \text{otherwise.} \end{cases}$$

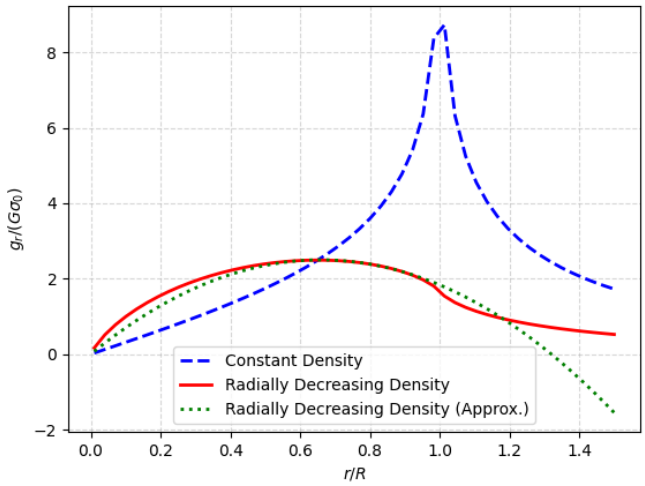


Figure 6.2: Magnitude of the radial component of the field strength due to a disk. The peak at $r = R$ for the constant density disk is, in fact, infinite.

6.1.3 Initial conditions

The total field $\mathbf{g} = \mathbf{g}_D + \mathbf{g}_H$ is used to find initial velocities for the particles with initial positions $(x, y, 0)$. The formula for the centripetal force yields

$$\frac{v^2}{r} = -g_r$$

and thus

$$\mathbf{v} = \left(-v\frac{y}{r}, v\frac{x}{r}, 0\right)$$

with $v = \sqrt{-rg_r}$ for counter-clockwise rotation.

6.1.4 Disk with a hole

While testing the implemented methods, we observed that for modeling systems comprising multiple galaxies, it may be beneficial to replace the fixed halo with a single massive particle. In such a case, the simulation turns out to be stable if there are no particles in close vicinity to the galaxy center. This means that the galactic disk ought to have a hole of radius r_0. Using an approach analogous to the one used in subsection 6.1.1, we obtain the following relations that can be used for sampling points (ϕ, r) in polar coordinates

$$\phi = 2\pi u \quad \text{and} \quad 2(r^3 - r_0^3) - 3R_D(r^2 - r_0^2) + (R_D - r_0)^2(2r_0 + R_D)u = 0,$$

where $u \sim U(0, 1)$.

Getting rid of the external halo field simplifies the treatment of complex systems significantly, as we do not need to keep track of the movement of the halo. The massive particle in the galaxy center, which effectively replaces it, can be treated just as any other particle in the simulation.

6.2 Globular cluster model

A globular cluster is a formation comprising a large number of stars, closely packed in a spherically symmetric form [14]. In this work, we employed the implemented methods to simulate such a structure using the Plummer model. The model was chosen due to its simplicity and abundance of dedicated resources.

In the Plummer model, the density of a cluster is given by

$$\rho(r) = \frac{3M}{4\pi a^2}\left(1 + \frac{r^2}{a^2}\right)^{-5/2}, \tag{6.2}$$

where the parameter a controls the spread of the distribution (the size of the cluster core) [1]. Hence, the particles are sampled from a distribution with PDF $p(r) = \rho(r)/M$. Similarly to the galaxy model, we use inverse transform sampling to initialize the positions of the particles. The marginal CDFs are easily calculated as

$$F_R(r) = \frac{r^3}{a^3}\left(1 + \frac{r^2}{a^2}\right)^{-3/2}, \quad F_\Theta(\theta) = \frac{1}{2}(1 - \cos\theta), \quad F_\Phi(\phi) = \frac{\phi}{2\pi},$$

which means that given a random variable $u \sim U(0, 1)$, we can generate random points with spherical coordinates

$$r = a(u^{-2/3} - 1)^{-1/2}, \quad \theta = \cos^{-1}(1 - 2u), \quad \phi = 2\pi u, \tag{6.3}$$

consistent with Equation 6.2 (assuming equal mass of all particles).

In the Plummer model, the gravitational potential is given by

$$\phi(r) = -\frac{GM}{\sqrt{r^2 + a^2}}.$$

This expression can be used to determine the escape velocity v_e at a distance r from the center of the system. Using the law of conservation of energy, we get

$$\frac{1}{2}v_e + \phi(r) = 0 \Rightarrow v_e = \sqrt{-2\phi(r)}.$$

For any r, the probability distribution of $q \equiv v/v_e$ is given by [1]

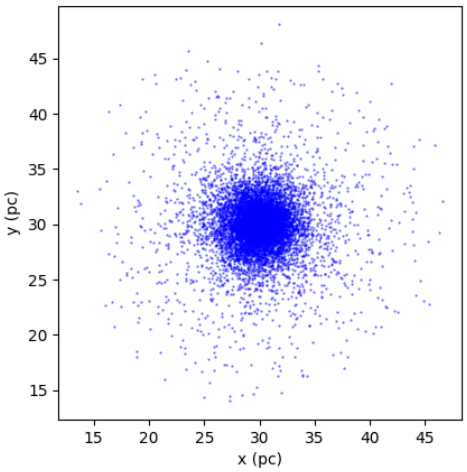
$$g(q) = Nq^2(1 - q^2)^{7/2}, \tag{6.4}$$

where N is a normalization constant, which can be calculated to be $N = 512/(7\pi)$. The CDF of the distribution in Equation 6.4, $G(q) = \int_0^q g(q')dq'$, can be determined using symbolic integration. The algebraic expression that one obtains in this way is lengthy, and we will not cite it here. The random values of q, consistent with the PDF in Equation 6.4, are again obtained using inverse transform sampling; the equation $G(q) = u$ is solved for q by finding the roots of $G(q) - u$ using Newton's method. Finally, the magnitude v of the velocity vector $\mathbf{v}$ is set to $v = qv_e$. The direction of $\mathbf{v} = (v_x, v_y, v_z)$ is chosen uniformly at random, i.e.

$$v_x = v\sin\theta\cos\phi, \quad v_y = v\sin\theta\sin\phi, \quad v_z = v\cos\theta,$$

where θ and ϕ are generated in the same way as in Equation 6.3.

The comparison of particle positions generated using the method above with an example of a real-world globular cluster is shown in Figure 6.3.



(a) Generated data ($M = 10^6 M_\odot$, $a = 2$ pc, $N = 10,000$).

(b) Real-world globular cluster (Messier 13).

Credit for Figure 6.3b: Giuseppe Donatiello.

Figure 6.3: The particle positions generated using the described model compared to a real-world globular cluster.

Chapter 7

Results

All implemented methods were applied to three scenarios:

1. Spiral galaxy simulation (section 6.1),
2. Globular cluster simulation (section 6.2),
3. Galaxy collision simulation (two galaxies modelled as "disks with holes", subsection 6.1.4).

The choice of these scenarios well illustrates the versatility of the methods introduced over the last couple of sections.

7.1 Spiral galaxy simulation

The parameters used in the simulation of a spiral galaxy are shown in Table 7.1. The galaxy is simulated

Parameter	**Value**
Halo radius	3 kpc
Halo mass	$60 \times 10^9 M_\odot$
Disk radius	15 kpc
Disk mass	$15 \times 10^9 M_\odot$
Disk thickness	0.3 kpc
Disk density profile	Uniformly decreasing

Table 7.1: Galaxy model parameters used in the simulation.

as an isolated system; however, in deriving Equation 2.5, periodic boundary conditions were assumed. The simplest way (and the one used) to obtain a free-space solution from the PM method is to extend the computational domain twice in every dimension and fill the space unused in mass distribution with zeros. The total size of the potential mesh used was $128 \times 128 \times 64$ with the region of interest occupying a box of size $60\,\text{kpc} \times 60\,\text{kpc} \times 30\,\text{kpc}$ located in a $64 \times 64 \times 32$ octant of the mesh.

7.1.1 Particle-mesh method

In the PM method, $N = 50{,}000$ particles were used. Cell size H and timestep length were set to $60/64 = 0.9375$ kpc and 1 Myr, respectively. For convenience, the full configuration of the PM method is presented in Table 7.2. The system's evolution over 200 Myr is shown in Figure 7.1.

Parameter	**Value**
Effective mesh size	$64 \times 64 \times 32$
H (cell size)	$60/64 = 0.9375$ kpc
DT (time step)	1 Myr
Mass assignment scheme	TSC
Finite difference	Two-point
Green's function	Derived from discretized Laplacian
Time integration method	Leapfrog
Number of particles	50,000

Table 7.2: PM method configuration.

During the simulation, total energy $E = \mathrm{KE} + \mathrm{PE}$, angular momentum $\mathbf{l}$, and the z-component of the momentum vector $\mathbf{p}$ should stay constant. The x- and y-components of momentum change due to the presence of an external gravitational field (representing the halo). We can verify if this variation satisfies the expected relation

$$\dot{\mathbf{p}} = \mathbf{F}^{\mathrm{ext}} \tag{7.1}$$

by finding the initial total momentum $\mathbf{p}(t = 0)$ and incrementing the value of $\mathbf{p}$ in each time-step by $\mathbf{F}^{\mathrm{ext}}\mathrm{DT}$.

The exact calculation of the potential energy [13] using the formula

$$\mathrm{PE} = -\sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{G m_i m_j}{r_{ij}}$$

is computationally infeasible considering the $O(N^2)$ cost. An approximation based on the potential values at mesh points,

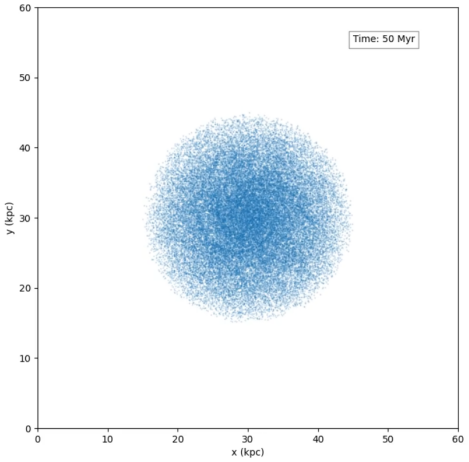
$$\mathrm{PE} \approx \frac{V}{2} \sum_{\mathbf{p}} \rho(\mathbf{x_p}) \phi(\mathbf{x_p}),$$

is used instead (for derivation, refer to [8]).

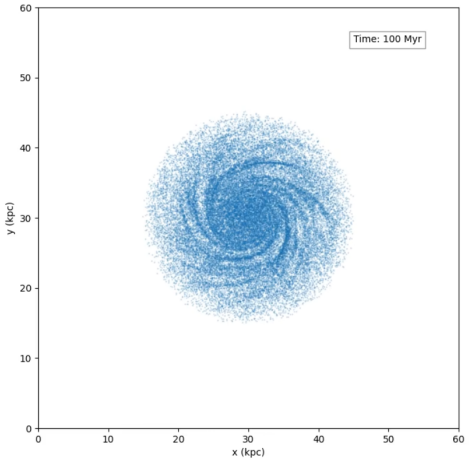
7.1.2 Particle-particle particle-mesh method

The $\mathrm{P^3M}$ based simulation uses the same parameters as the PM method. The reference force was calculated using the S_1 shape formula (Equation 3.2) with particle diameter $a = 3H$. The cutoff radius was set to $r_e = 0.7a$. One extra free parameter is the *softening length* ϵ, which modifies the universal law of gravitation so that division by zero can be avoided, i.e., the modified law is

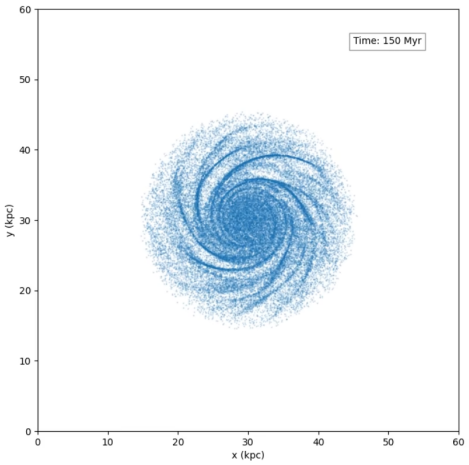
$$F_{ij}^{\mathrm{soft}}(r) = \frac{G m_i m_j}{r_{ij}^2 + \epsilon^2}.$$



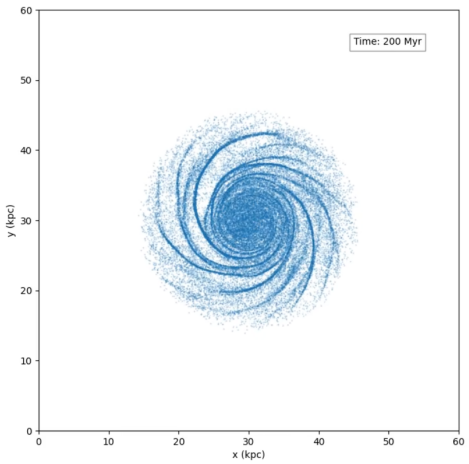
(a) $t = 50\,\text{Myr}$



(b) $t = 100\,\text{Myr}$



(c) $t = 150\,\text{Myr}$



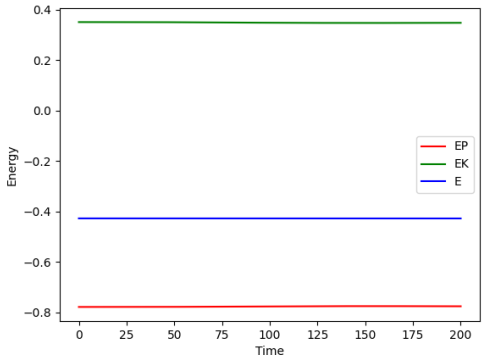
(d) $t = 200\,\text{Myr}$

Figure 7.1: Evolution of a spiral galaxy as predicted by the PM method.

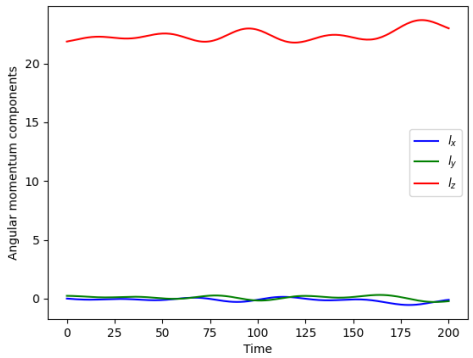
In the simulation, ϵ was set arbitrarily to 1.5 kpc. The whole configuration of the P^3M method is conveniently shown in Table A.1. The system's evolution is presented in Figure A.1, and the graphs of energy, angular momentum, and momentum components vs. time are shown in Figure A.2.

7.1.3 Barnes-Hut algorithm

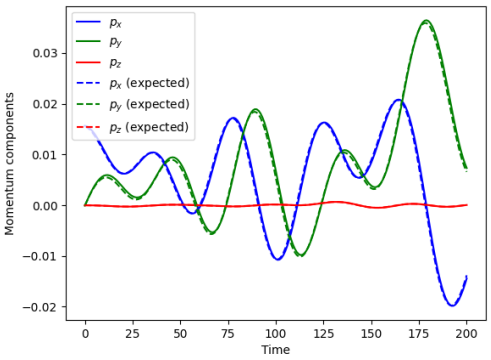
For the Barnes-Hut algorithm test, the initial conditions of the system remain the same as in the two previous simulations. The softening length ϵ was set to 1 kpc, and quadrupole terms were omitted. The whole configuration of the method is shown in Table A.2. The snapshots of the simulation are displayed in Figure A.3, and the graphs showing how the energy, momentum, and angular momentum varied over time are shown in Figure A.4.



(a) Energy



(b) Angular momentum



(c) Momentum; broken lines represent the expected momentum following Equation 7.1

Figure 7.2: Fundamental physical quantities describing the system over time in the PM simulation. Time is in Myr, and the quantities are expressed in units consistent with Table 7.1

7.1.4 Commentary

All of the implemented methods successfully reproduced the expected structural features of a spiral galaxy. A region of high density is clearly localized in the center of the galaxy, and characteristic spiral arms emerge around the 150 Myr mark. The snapshots from all three simulations (Figure 7.1, Figure A.1, and Figure A.3) show similar evolution of the system over time, with the last snapshot of each simulation visually resembling a real-world spiral galaxy (cf. Figure 6.1b). It is worth noting, however, that the system evolves more rapidly in the PM-based simulation. For instance, the spiral arms are significantly more developed at $t = 100$ Myr (Figure 7.1b) compared to the same time point in the simulations using the other two methods. This accelerated evolution is likely due to the smaller "induced softening length" inherent to the PM method, which results in stronger gravitational forces. These limitations were anticipated and discussed in subsection 2.6.2.

Energy plots (Figure 7.2a, Figure A.2a, and Figure A.4a) show that each of the methods tested conserved the total energy. The plots of angular momentum over time (Figure 7.2b, Figure A.2b, and Figure A.4b) illustrate that the angular momentum was *on average* constant with the variations stemming

from the presence of the external halo field. Finally, the momentum plots (Figure 7.2c, Figure A.2c, and Figure A.4c) demonstrate that the z-component of the momentum vector remains constant, while the remaining components change (again due to the external field). The comparison of the actual values of the momentum with the theoretical values, updated in each timestep according to Equation 7.1, shows that the change in the x- and y-components is correct.

7.2 Globular cluster simulation

The globular cluster is simulated using the Plummer model with parameters set to the values in Table 7.3. The *maximum radius* parameter was introduced to confine the cluster to a predefined computational

Parameter	**Value**
a (spread)	2 pc
Mass	$1 \times 10^6 M_\odot$
Maximum radius	15 pc

Table 7.3: Galaxy model parameters used in the simulation.

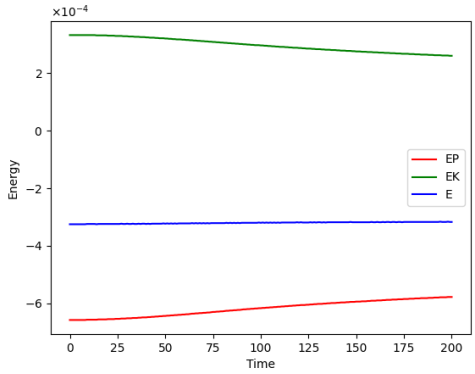
domain. We restrict the presentation of our results to the simulation conducted using the Barnes-Hut algorithm since other methods yielded practically the same results.

The configuration of the Barnes-Hut algorithm used in the globular cluster simulation is given in Table 7.4. The conservation of energy, momentum, and angular momentum during the simulation is

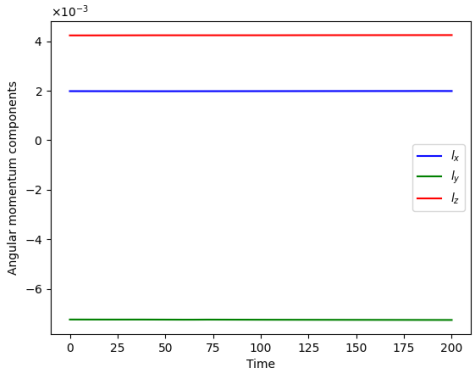
Parameter	**Value**
θ (opening angle)	0.5
ϵ (softening length)	0.5 pc
DT (time step)	1 kyr
Highest multipole term	quadrupole

Table 7.4: Barnes-Hut method configuration.

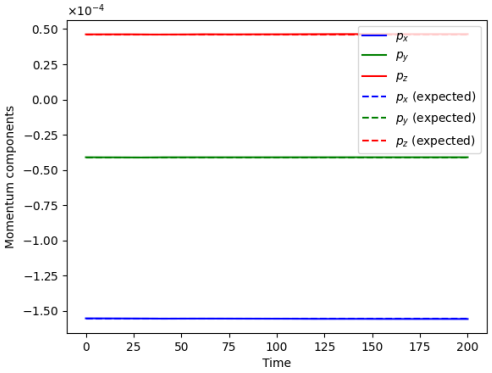
demonstrated in Figure 7.3. Notice that the conservation of momentum and angular momentum is possible due to a lack of any external forces (this is contrasted with the previous test – galaxy simulation with halo modeled a fixed, external field). The evolution of the system (initial positions and after 200 kyr) is shown in Figure 7.4. As can be seen, the system remains stable and bound gravitationally. Also, in the initial snapshot (Figure 7.4a), the effect of restricting the sampled points to a sphere of radius can be seen. By the end of the simulation, the system bears a general resemblance to real-world globular clusters (cf. Figure 6.3b).



(a) Energy

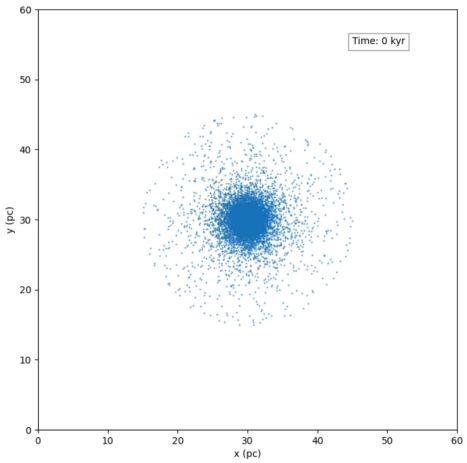


(b) Angular momentum

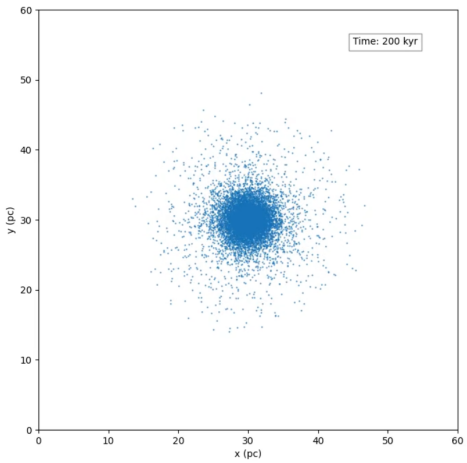


(c) Momentum; broken lines represent the expected momentum following Equation 7.1

Figure 7.3: Fundamental physical quantities describing the system over time in the Barnes-Hut algorithm. Time is in kyr, and the quantities are expressed in units consistent with Table 7.3



(a) $t = 0\,\text{kyr}$



(b) $t = 200\,\text{kyr}$

Figure 7.4: Evolution of a globular cluster as predicted by the Barnes-Hut algorithm.

7.3 Galaxy collision simulation

We conclude by showcasing the applications of our program with a simulation of the collision of two galaxies. The parameters describing the galaxies are given in Table 7.5.

Parameter	**Galaxy 1**	**Galaxy 2**
Center position	(40, 30, 15) kpc	(80, 30, 15) kpc
Halo radius	3 kpc	2 kpc
Halo mass	$60 \times 10^9 M_\odot$	$40 \times 10^9 M_\odot$
Disk radius	15 kpc	10 kpc
Disk mass	$15 \times 10^9 M_\odot$	$10 \times 10^9 M_\odot$
Disk thickness	0.3 kpc	0.3 kpc
Disk density profile	Uniformly decreasing	Uniformly decreasing
Number of particles	3×10^4	3×10^4

Table 7.5: Galaxy model parameters used in the simulation.

7.3.1 Barnes-Hut algorithm

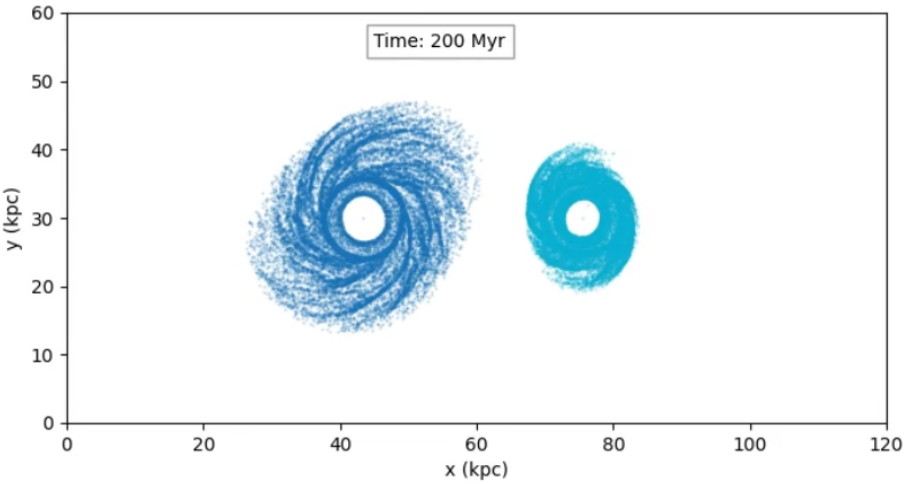
The configuration of the algorithm is given in Table 7.6. The course of the collision is shown in Figure 7.5.

Parameter	**Value**
θ (opening angle)	1
ϵ (softening length)	1.3 kpc
DT (time step)	1 Myr
Simulation duration	400 Myr
Highest multipole term	Quadrupole

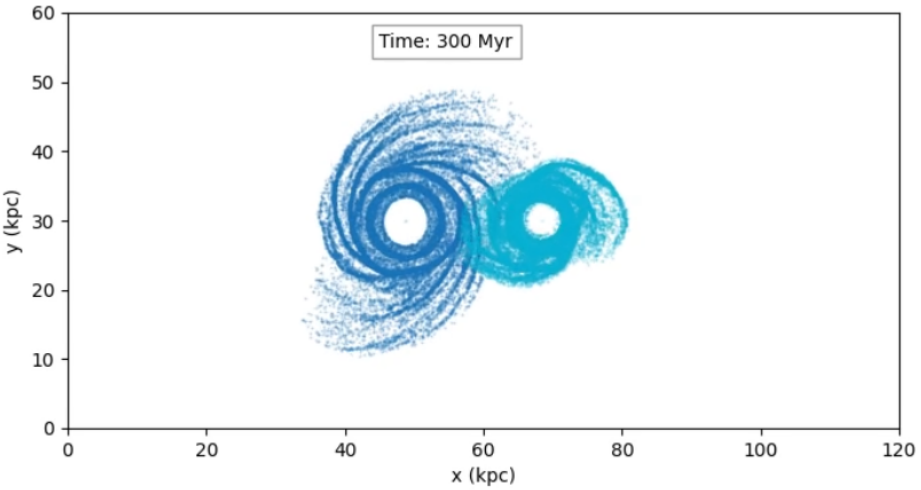
Table 7.6: Barnes-Hut method configuration.

Plots of momentum, angular momentum, and energy are shown in Figure 7.6.

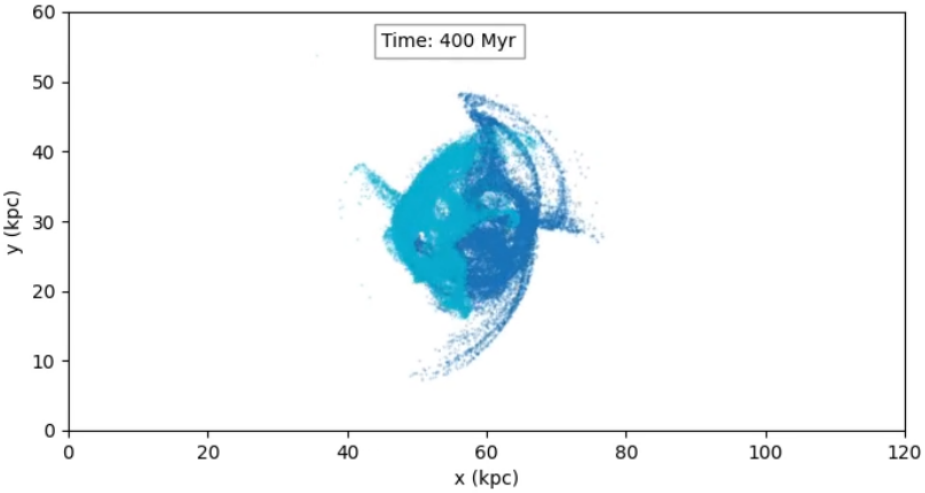
As shown in Figure 7.6a, the total energy remains nearly constant until the centers of the two galaxies approach each other closely. At this stage, substantial numerical errors are likely introduced, resulting in a sharp discontinuity between the $t = 350$ Myr and $t = 400$ Myr timestamps. Around this point, the angular momentum also ceases to be conserved. Interestingly, the system's momentum oscillates around the expected value throughout the simulation, with the x-component displaying the most pronounced deviations. This behavior reflects the fact that the Barnes-Hut algorithm does not strictly conserve momentum, as the interparticle forces calculated in the Barnes-Hut scheme do not follow Newton's third law. We conclude by pointing out that the violation of the laws of physics during the collision renders the results collected after this point unreliable.



(a) $t = 200\,\text{Myr}$

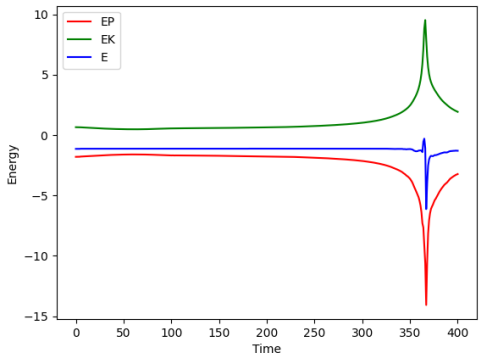


(b) $t = 300\,\text{Myr}$

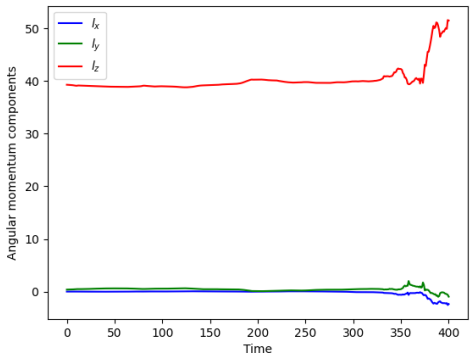


(c) $t = 400\,\text{Myr}$

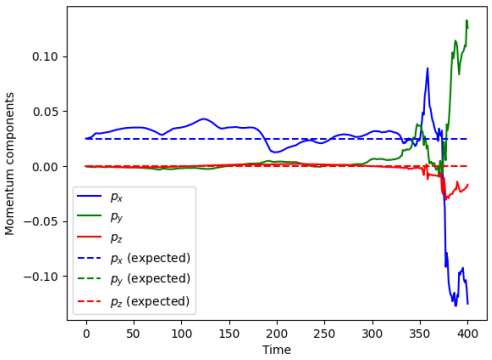
Figure 7.5: Galaxy collision stage (Barnes-Hut algorithm).



(a) Energy



(b) Angular momentum



(c) Momentum; broken lines represent the expected momentum following Equation 7.1

Figure 7.6: Fundamental physical quantities describing the system over time in the Barnes-Hut algorithm. Time is in Myr, and the quantities are expressed in units consistent with Table 7.1

7.3.2 P3M method

The configuration of the algorithm is the same as in the case of a single galaxy (Table A.1). Plots of the physical quantities under consideration are displayed in Figure A.5, and snapshots from the simulations are shown in Figure A.6. The figures show that the momentum is conserved for the entire duration of the simulation (Figure A.5c) and that there is a slight jump in the components of the angular momentum between 300 and 400 Myrs (Figure A.5b), corresponding to the galaxy centers coming closely together. The energy plots (Figure A.5a) roughly match the energy plots in the Barnes-Hut method (Figure 7.6a). However, the potential energy (and, as a consequence, the total energy) oscillates irregularly during the simulation. A probable source of this phenomenon is the inaccuracy in representing the very deep potential well generated by the massive particle in the center of the galaxy by the potential mesh. This leads to fluctuations as the massive particle moves across the grid. However, a more in-depth study would need to be undertaken to find a definitive explanation.

7.3.3 PM method

The method's configuration remains the same as in the single galaxy simulation (Table 7.2). Plots of the physical quantities under consideration are displayed in Figure A.7, and snapshots from the simulations are shown in Figure A.8. The deviations of the momentum and angular momentum components from their initial values (Figure A.7b and Figure A.7c) are greater compared to the P3M method. Additionally, the oscillations in the potential and total energies are significant, raising serious doubts about whether the simulation can be accepted as physically valid. Examination of the system at $t = 200$ and 300 Myr (Figure A.8a and Figure A.8b) reveals that the interactions between the galactic center and the surrounding disk particles are not accurately captured. The central 'hole" in the disk begins to vanish as nearby particles collapse inward. In contrast, the Barnes-Hut (Figure 7.5a, Figure 7.5b) and P^3M (Figure A.6a, Figure A.6b) simulations at the same time points maintain a well-defined circular hole.

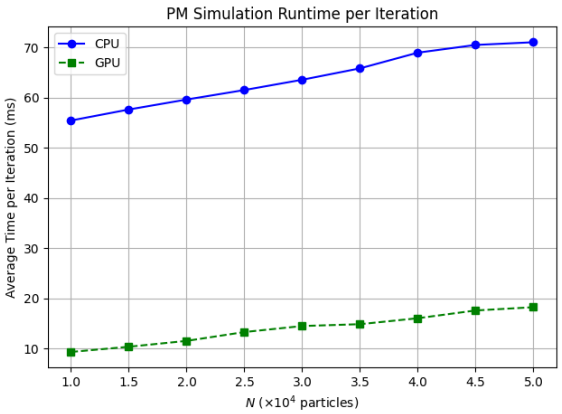
7.4 Performance analysis

The performance analysis was conducted in the setting of the spiral galaxy simulation on a computer equipped with an Intel Core i7-9750H CPU (2.60GHz) and an NVIDIA GeForce GTX 1650 GPU. By that, we mean that not only the particle distribution was the same as in section 7.1, but also the configuration of the algorithms remained unchanged. Moreover, to obtain results that most closely correspond to a real scenario of using the program, diagnostic information collection was enabled, and the simulation state was saved after each frame. The diagnostic information mentioned here refers collectively to energy (kinetic, potential, and total), momentum, and angular momentum. It is essential to keep in mind that in the case of the PM and P^3M methods, recording these quantities comes with an additional overhead of converting the state of the system from code to "original" units. Potentially, suitable code units could also be introduced in the case of the Barnes-Hut algorithm, but we did not find this necessary. Saving the state of the system in each frame also incurs additional overhead related to disk I/O operations. In our implementation, the output is buffered so that data is written to disk at a reasonable frequency. We also note that the reported running time is averaged over 100 iterations.

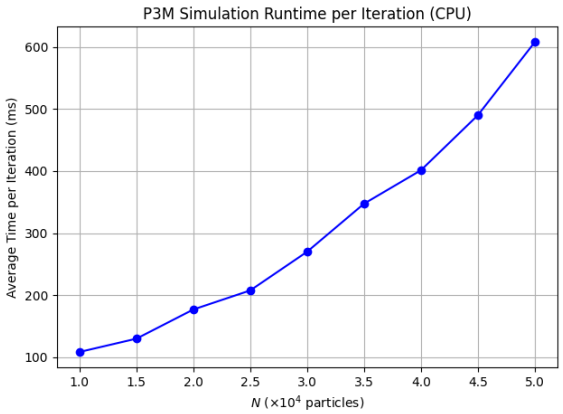
The running time for the PM method is shown in Figure 7.7. The GPU implementation provides, at best, a five-fold speedup over the CPU implementation. This is in stark contrast with the results reported in subsection 2.7.2 (1200% speedup). The difference between the two results stems from the fact in subsection 2.7.2, we considered an "idealized scenario" where the data transfers from device to host and disk I/O operations were reduced to the absolute minimum, saving only the *final state* of the simulation. Both plots (GPU and CPU variants) exhibit linear time dependence as expected.

The P^3M method comes out as the least performant method among these considered in the thesis. The runtime per iteration was up to 8.5 times longer than the CPU variant of the PM method and up to 3.5 times longer than the Barnes-Hut algorithm in its standard variant. Moreover, it is apparent that the algorithm does not scale linearly but, we pointed out in subsection 3.1.4, the influence of the $O(N^2)$ direct summation part of the algorithm can reduced at the expense of weaker accuracy.

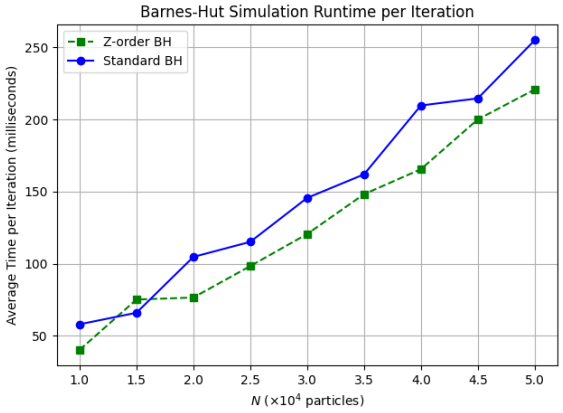
The Barnes-Hut algorithm was timed in two configurations: with the standard tree construction and



(a) PM method



(b) P^{3}M method



(c) Barnes-Hut algorithm

Figure 7.7: Running time of the different methods as a function of the number of particles.

the improved one (introduced in section 4.3). The range of N considered in the test is too small to reveal the linearithmic runtime dependence on N fully. In any case, the Barnes-Hut algorithm comes second in the comparison, with running times 3.5 slower than the PM method in the standard variant and only 3.2 times slower if the improved tree construction is used. We note that a comparison with Figure 4.3 reveals that the tree construction is responsible for around 15% of the total running time of the simulation.

Chapter 8

Conclusions

This thesis discussed three classic methods of N-body simulations: PM, P^3M, and Barnes-Hut.

8.1 Summary of key findings

We introduced the PM method by focusing on its key aspects: mass assignment, Poisson's equation solver, gradient approximation for field calculation, and interpolation. This systematic description led us to observe the connection between the Fourier transform and the diagonalization of the Laplacian operator. This connection, in turn, clarified the derivation of the Green's function for the so-called "Poor Man's" Poisson solver.

The error analysis of the PM method of the field due to a single mass showed that the best results were obtained using the TSC scheme for mass assignment and field interpolation, which confirmed our expectations. The global error analysis, on the other hand, yielded results which at first seemed at odds with the preceding theoretical developments – the NGP scheme seemed to have performed best by giving results closest to the PP method. This conclusion was refuted based on an observation that the forces generated by the PM method are "inherently softened". Indeed, a comparison with softened PP forces spoke in favor of the higher-order schemes (CIC and TSC).

We found that our GPU implementation is up to 12 times faster compared to the multithreaded CPU implementation but only if data transfers between the CPU and GPU and disk I/O are limited; when these conditions are not met, the speedup is more modest – only five-fold. We also investigated if additional performance gains are possible if the data needed for the gradient calculation is first fetched into shared memory but no such gain was observed.

Turning to the P^3M method, our focus was twofold: designing an optimal Green's function that closely matches a given reference force and the short-range correction.

The error analysis conducted in the first area showed that in our implementation of the PM method, the mesh force correctly reproduced the reference force once the appropriate Green's function was used. Interestingly, our analysis showed that using the S_1 shape gives smaller errors (difference between mesh- and reference force) compared to the S_2 shape. We could not find any similar study in the literature,

however, that could confirm if our observation is valid. Our single-source analysis confirmed that the TSC scheme minimizes the force approximation error, as expected.

We proposed a simple lock-free parallelization scheme of the short-range correction procedure which yielded a four-fold speedup. We consider this result satisfactory although further optimization of the current implementation of the P^3M method is highly desirable. Confusingly, the optimization proposed in [8] (sorting HOC lists) did not improve the performance and produced the opposite effect. We confirmed that sorting the HOC lists leads in a lot of cases to early returns from the short-range correction loop, however, this does not seem to offset the added cost of sorting the lists. Finally, the global error analysis confirmed that higher-order schemes lead to smaller approximation errors and that the error decreases with increasing cutoff radius and particle diameter, as we expected. We found that the global error can be reduced beyond 1% if $a \geq 4H$.

Finally, we examined the Barnes-Hut algorithm, with particular attention to its force approximation accuracy and computational efficiency. Our study of the Barnes-Hut algorithm showed that the approximation error is minimized for small values of the opening angle θ and that the running time of the algorithm decreases with increasing values of θ – both results were to be expected. Additionally, we confirmed that including the quadrupole moment in the force calculation leads to a significant reduction in the error of force approximation. We also found that the added cost due to including the quadrupole moment is negligible and that the error can be brought down to as little as 10^{-5}. Interestingly, $\theta = 0$ did not cause the error to vanish completely, likely due to numerical errors.

We attempted to improve the performance of the Barnes-Hut algorithm by accelerating the tree construction step. For that, we used the approach based on Z-ordering the particles and inserting them at the last-visited node in the tree with the hope that this will lead to better cache utilization. The approach turned out to be successful and allowed us to achieve around 40% speedup over the standard tree-construction algorithm which translated into 13% speedup of the whole simulation.

The implemented methods were tested in three scenarios: a simulation of a spiral galaxy, a globular cluster, and a collision of two galaxies.

The first scenario did not demonstrate the superiority of any particular method. All methods yielded similar evolution of the spiral galaxy and the plots of energy, momentum, and angular momentum were practically the same for all methods.

Similarly, the globular cluster simulation gave similar results for all tested methods.

In the galaxy collision simulation scenario, the P^3M method yielded the most physical results, conserving angular momentum and momentum. The energy, however, oscillated violently around the expected values, and so was only *on average* constant.

In conclusion, the P^3M and Barnes-Hut codes provided the best results, with the Barnes-Hut algorithm being faster than P^3M in the tested scenarios. At the same time, the Barnes-Hut algorithm has problems with momentum conservation but these problems could probably be mitigated by including higher multipole terms. Overall, this work demonstrates the feasibility and trade-offs of classical N-body methods for different astrophysical simulations, laying the groundwork for future improvements in both accuracy and computational performance.

8.2 Future work

The program created for this thesis is by no means complete. There are at least a few improvements and areas to investigate that we would like to attend to in the future:

1. Implement force sharpening (see subsection 2.6.2) and investigate its influence on the accuracy of the PM method.

2. Investigate if there is an improvement in the performance of the GPU implementation of the PM method if shared memory is used in the four-point finite difference variant. We admittedly verified that there is no gain in the case of the two-point difference, but since more global memory reads are necessary for the four-point variant, some improvement should be expected.

3. More work is needed to make the P^3M up to par with performance-wise. Verify why sorting the linked lists did not bring an expected improvement in the running time.

4. GPU implementation of the Barnes-Hut algorithm is possible and reportedly achieves ten-fold speedups over the CPU implementation (see [4]). Extending the program by adding this functionality would be a great improvement.

5. Currently the data visualization is handled by a number of Python scripts that load the data produced by the main C++ program. This quite often leads to annoying errors when the format of the data produced by the main program is changed and becomes out of sync with the scripts.

Bibliography

[1] S. J. Aarseth, M. Henon, and R. Wielen. A comparison of numerical methods for the study of star cluster dynamics. *Astronomy and Astrophysics*, 37(1):183–187, December 1974.

[2] Martin Aigner and Günter M. Ziegler. *Proofs from THE BOOK*. Springer, Berlin, Germany, 6 edition, 2018.

[3] E. Athanassoula, E. Fady, J. C. Lambert, and A. Bosma. Optimal softening for force calculations in collisionless n-body simulations. *Monthly Notices of the Royal Astronomical Society*, 314(3):475–488, 05 2000.

[4] Martin Burtscher and Keshav Pingali. Chapter 6 - an efficient cuda implementation of the tree-based barnes hut n-body algorithm. In Wen mei W. Hwu, editor, *GPU Computing Gems Emerald Edition*, Applications of GPU Computing Series, pages 75–92. Morgan Kaufmann, Boston, 2011.

[5] Laurent Demanet, Jacob White, and Richard Zhang. Fourier-based fast methods for ordinary differential equations. `https://web.archive.org/web/20240802094957/https://math.mit.edu/icg/resources/teaching/18.336-spring2013/Notes_Set2.pdf`, February 2013. Lecture notes for 18.336/6.335: Fast Methods for Partial Differential and Integral Equations, MIT, February 7–14.

[6] I. M. Gelfand and G. E. Shilov. *Generalized Functions, Vol. 1: Properties and Operations*. Academic Press, 1964. Originally published in Russian; this volume introduces the theory of generalized functions and includes topics such as Fourier transforms, homogeneous distributions, and distributions on submanifolds.

[7] Lars Hernquist. Performance characteristics of tree codes. *Astrophysical Journal Supplement Series*, 64:715, August 1987.

[8] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. CRC Press, 1st edition, 1988.

[9] Andrey Kravtsov. Writing a pm code, March 2002. Accessed: 2025-04-03.

[10] Kenneth L. Nordtvedt, James E. Faller, and Alan H. Cook. Gravity. `https://www.britannica.com/science/gravity-physics`, 2025. Encyclopedia Britannica, Published May 30, 2025. Accessed June 6, 2025.

[11] NVIDIA Corporation. *CUDA C++ Programming Guide*. NVIDIA Corporation, release 12.9 edition, may 2025. Last accessed May 29, 2025.

[12] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3rd edition, 2007.

[13] John R. Taylor. *Classical Mechanics*. University Science Books, 2005.

[14] The Editors of Encyclopaedia Britannica. Globular cluster. `https://www.britannica.com/science/globular-cluster`, January 2024. Encyclopaedia Britannica.

[15] M. Trenti and P. Hut. Gravitational n-body simulations, 2008.

[16] Michele Trenti and Piet Hut. N-body simulations (gravitational). *Scholarpedia*, 3(5):3930, 2008. Revision #91544, Curator: Piet Hut.

[17] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Supercomputing '93, pages 12–21, New York, NY, USA, 1993. Association for Computing Machinery.

[18] J. Weiss. Certain aspects of the gravitational field of a disk. *Applied Mathematics*, 9:1360–1377, 2018.

[19] Eric W. Weisstein. Plane-filling function. `https://mathworld.wolfram.com/Plane-FillingFunction.html`, n.d. From MathWorld–A Wolfram Web Resource.

[20] Kelly Young. Andromeda galaxy hosts a trillion stars. *New Scientist*, May 2006. Accessed: 2025-05-25.

[21] Peter Young. Leapfrog method and other "symplectic" algorithms for integrating newton's laws of motion, 2019. Physics 115/242 Course Notes, UC San Diego.

[22] Tianchi Zhang, Shihong Liao, Ming Li, and Liang Gao. The optimal gravitational softening length for cosmological n-body simulations. *Monthly Notices of the Royal Astronomical Society*, 487(1):1227–1232, May 2019.

Appendix A

Simulations

A.1 Galaxy Simulations

A.1.1 P3M method

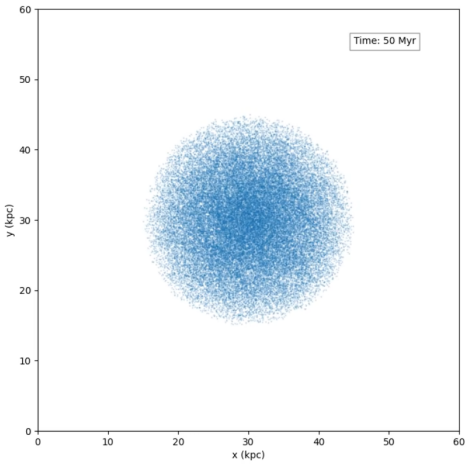
Parameter	**Value**
Effective mesh size	$64 \times 64 \times 32$
H (cell size)	$60/64 = 0.9375$ kpc
DT (time step)	1 Myr
Mass assignment scheme	TSC
Finite difference	Two-point
Time integration method	Leapfrog
Number of particles	50,000
a (particle diameter)	$3H$
Particle shape	S_1
r_e (cutoff radius)	$0.7a$

Table A.1: P3M method configuration.

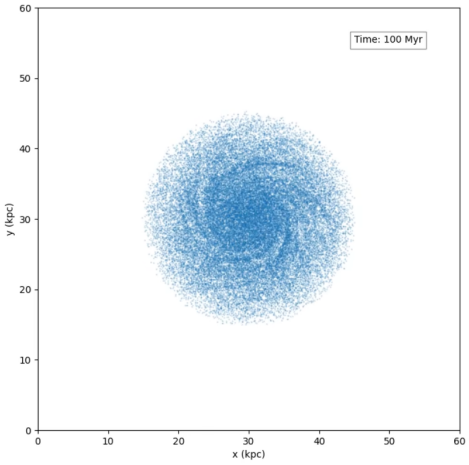
A.1.2 Barnes-Hut algorithm

Parameter	**Value**
θ (opening angle)	1
ϵ (softening length)	1 pc
DT (time step)	1 Myr
Highest multipole term	monopole

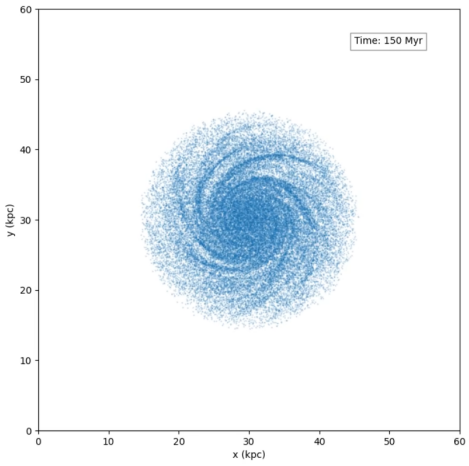
Table A.2: Barnes-Hut method configuration.



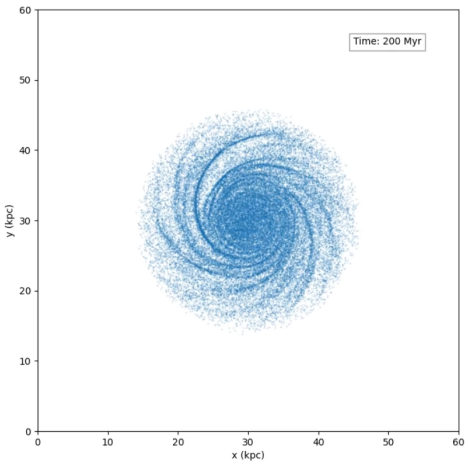
(a) $t = 50\,\mathrm{Myr}$



(b) $t = 100\,\mathrm{Myr}$

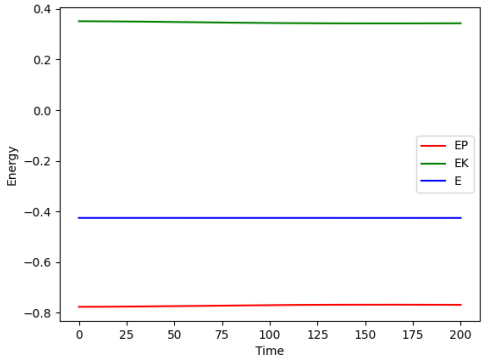


(c) $t = 150\,\mathrm{Myr}$

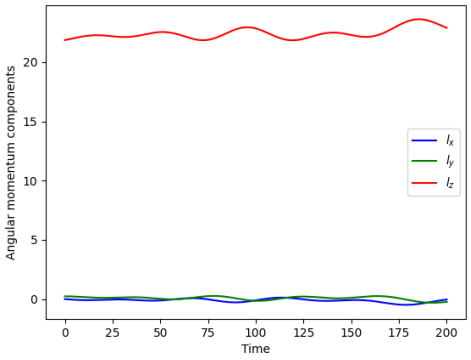


(d) $t = 200\,\mathrm{Myr}$

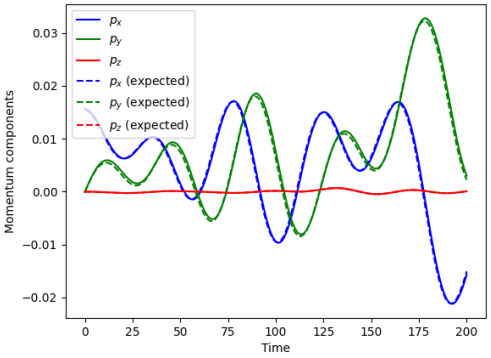
Figure A.1: Evolution of a spiral galaxy as predicted by the $\mathrm{P^3M}$ method.



(a) Energy

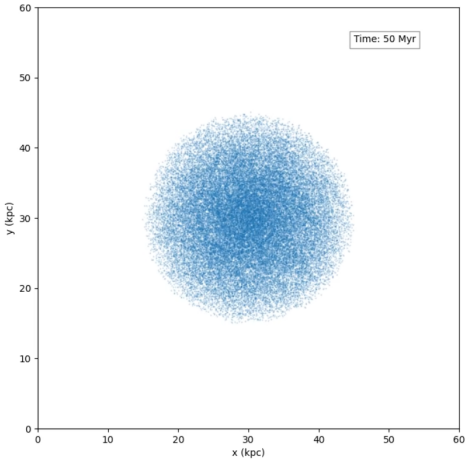


(b) Angular momentum

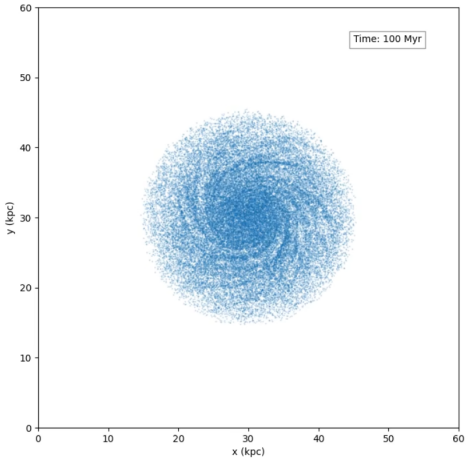


(c) Momentum; broken lines represent the expected momentum following Equation 7.1

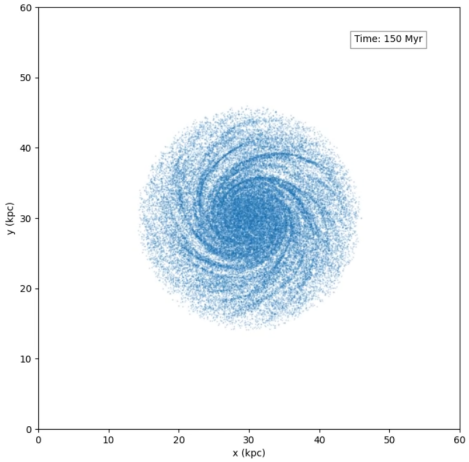
Figure A.2: Fundamental physical quantities describing the system over time in the P^3M simulation. Time is in Myr and the quantities are expressed in units consistent with Table 7.1



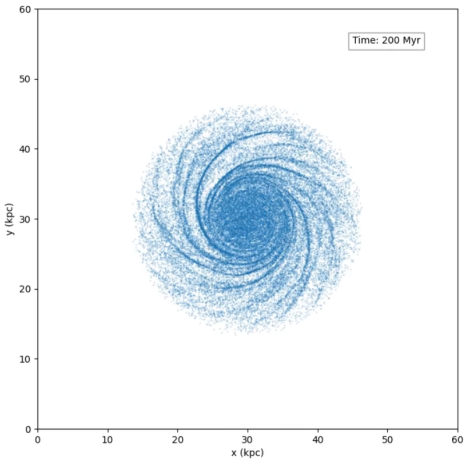
(a) $t = 50\,\mathrm{Myr}$



(b) $t = 100\,\mathrm{Myr}$

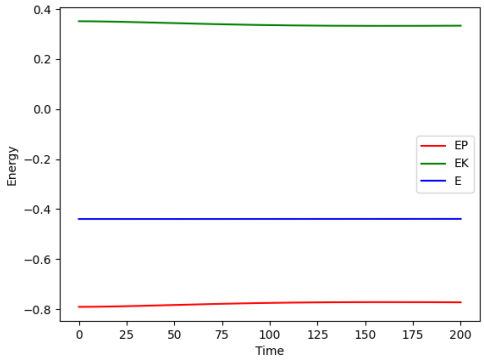


(c) $t = 150\,\mathrm{Myr}$

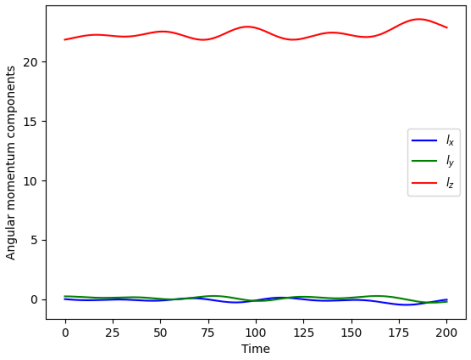


(d) $t = 200\,\mathrm{Myr}$

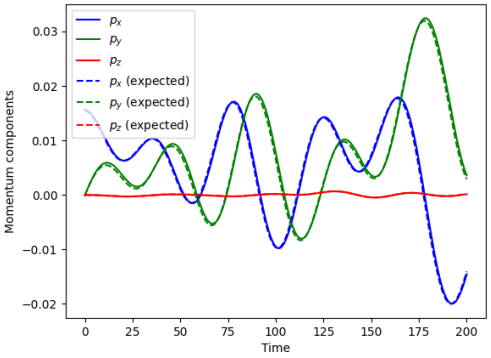
Figure A.3: Evolution of a spiral galaxy as predicted by the Barnes-Hut algorithm.



(a) Energy



(b) Angular momentum

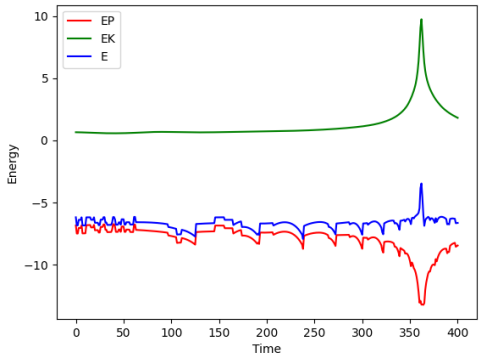


(c) Momentum; broken lines represent the expected momentum following Equation 7.1

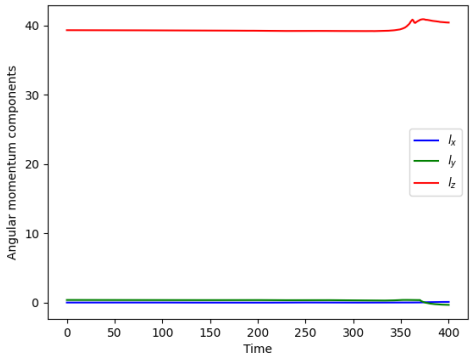
Figure A.4: Fundamental physical quantities describing the system over time in the Barnes-Hut algorithm. Time is in Myr and the quantities are expressed in units consistent with Table 7.1

A.2 Galaxy collision

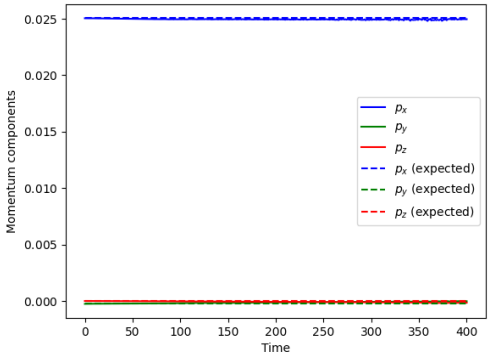
A.2.1 P3M method



(a) Energy



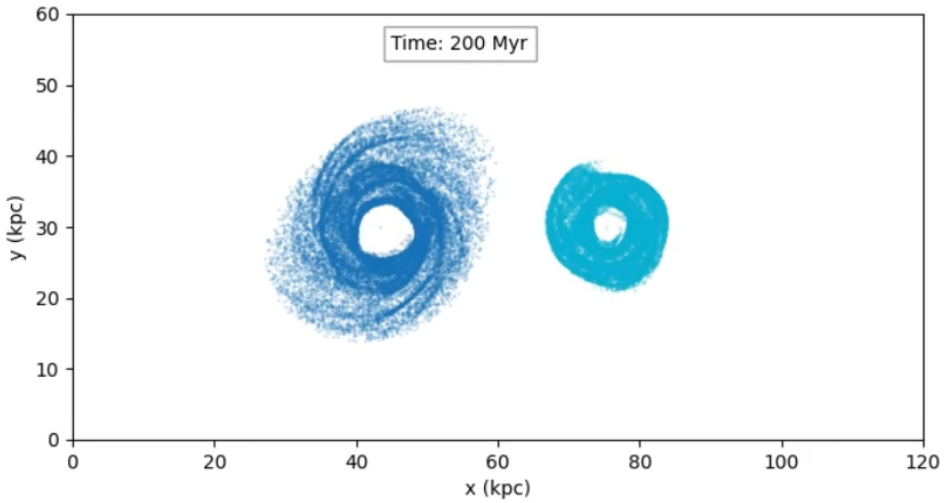
(b) Angular momentum



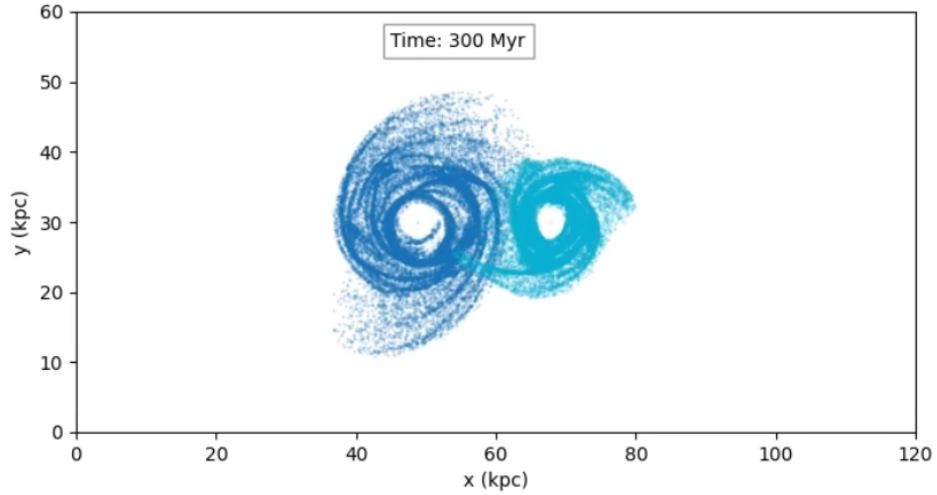
(c) Momentum; broken lines represent the expected momentum following Equation 7.1

Figure A.5: Fundamental physical quantities describing the system over time in the Barnes-Hut algorithm. Time is in Myr and the quantities are expressed in units consistent with Table 7.1

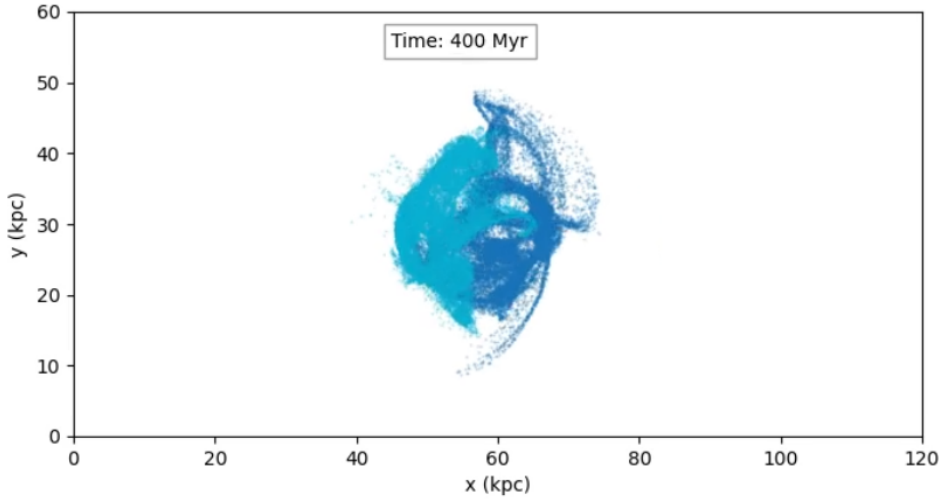
A.2.2 PM method



(a) $t = 200\,\mathrm{Myr}$

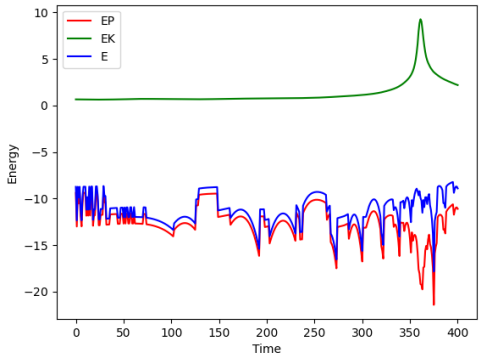


(b) $t = 300\,\mathrm{Myr}$

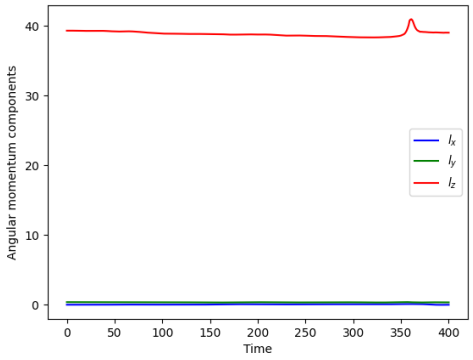


(c) $t = 400\,\mathrm{Myr}$

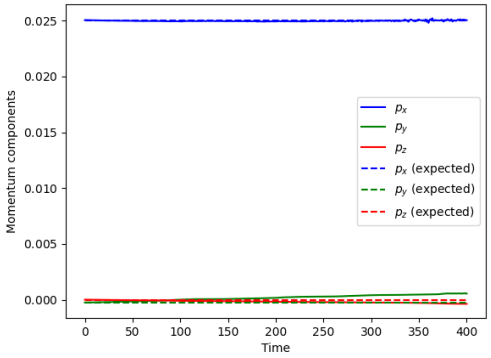
Figure A.6: Galaxy collision stage (P3M method).



(a) Energy

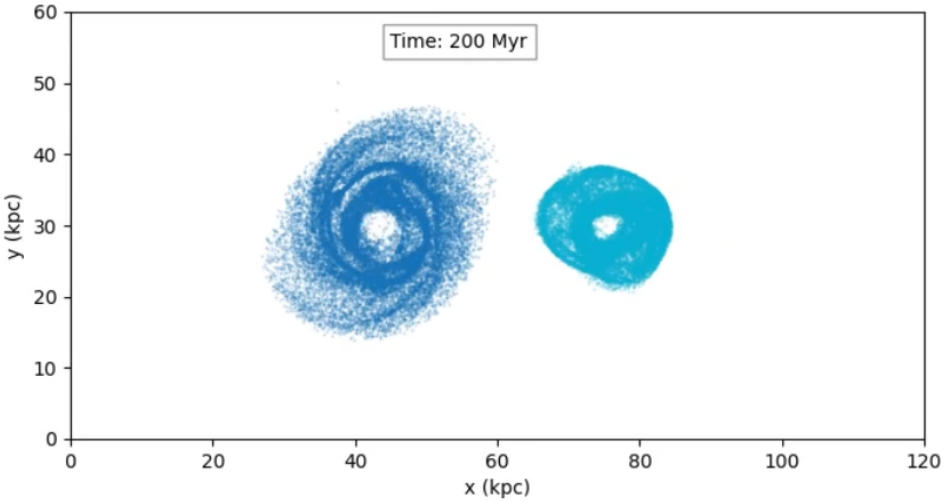


(b) Angular momentum

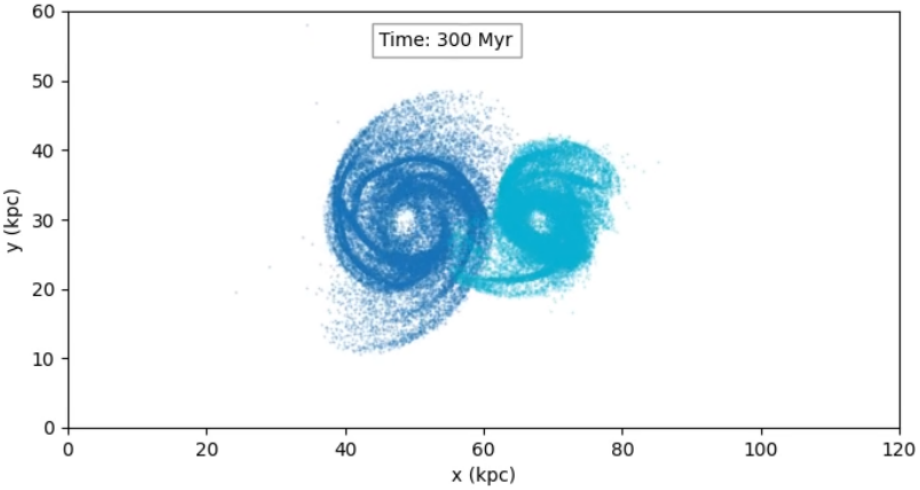


(c) Momentum; broken lines represent the expected momentum following Equation 7.1

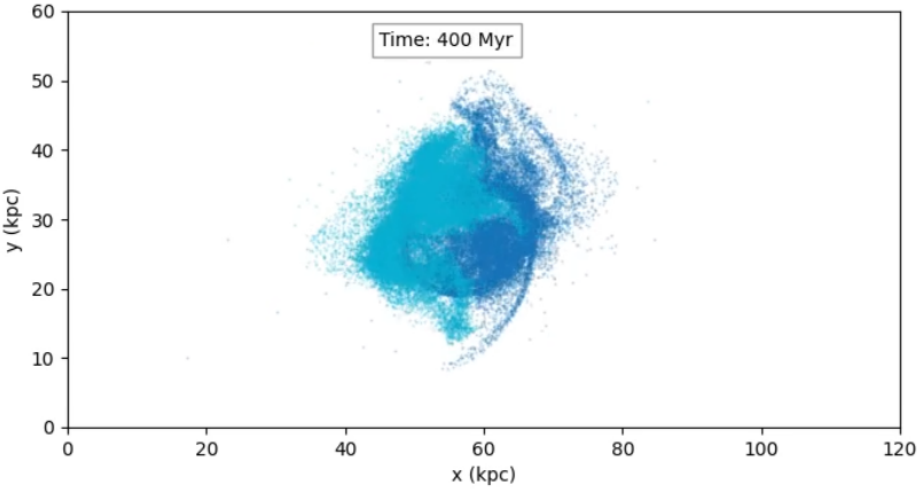
Figure A.7: Fundamental physical quantities describing the system over time in the Barnes-Hut algorithm. Time is in Myr and the quantities are expressed in units consistent with Table 7.1



(a) $t = 200\,\mathrm{Myr}$



(b) $t = 300\,\mathrm{Myr}$



(c) $t = 400\,\mathrm{Myr}$

Figure A.8: Galaxy collision stage (PM method).