School of Computer Science

College of Engineering and
Computer Science

The Fast Multipole Algorithm vs the Particle Mesh Ewald method

Joshua Nelson - u4850020

COMP3006 - Computer Science Research Project

Supervisor: Dr Eric McCreath

October 18, 2012

Abstract

The N body problem is common across the fields of physics, biology and chemistry. The classic solution to this problem has an inhibitive complexity in the class $O(n^2)$. Two alternative methods were examined: The Fast Multipole Algorithm, and the Particle Mesh Ewald Method, with better complexities of $O(n)$ and $O(n\log(n))$, respectively. These algorithms were implemented in Java, and their efficiencies were discussed and compared. The algorithms were run over typical molecular dynamics simulations to determine the most efficient algorithm for the N-body problem.

Contents

Chapter 1

Introduction

1.1 The N body problem

Suppose we have a collection of n bodies in some space, that interact with each other. Each body interacts with every other body in the system in a pairwise way. Often this pairwise interaction is a function of the distance between the bodies, and their properties, such as mass or electric charge. The task is to calculate the total effect on each body from every other body.

The N body problem is key to the simulation of many different scientific environments. The bodies may be astrophysical objects, such as planets or galaxies, interacting based on distance and body mass [6], or atoms in a molecular dynamics simulation, based on distance and particle charge.[9]. For the remainder of the report, we will discuss the N body problem in regards to Molecular dynamics, however the approaches can be generalised.

Chapter 2

Algorithms for the n body problem

2.1 The $O(n^2)$ solution

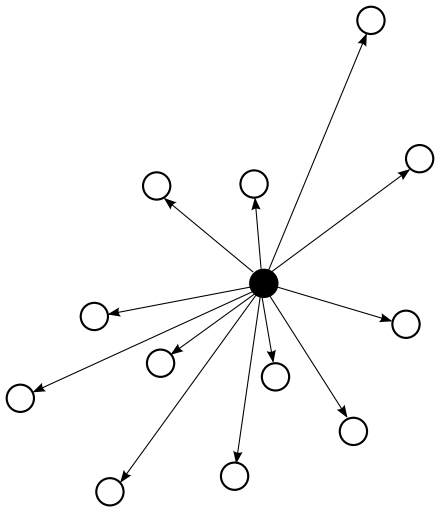

Figure 2.1: The naïve approach to the n body problem - calculate each interaction for each particle

The simplest solution to the N body problem is the basic $O(n^2)$ approach of calculating each interaction directly. Pseudocode for the algorithm is given below.

Data: r_i: particle positions, q_i: particle charges, N: number of particles, Q: output array of charges
for *i=0 to N* **do**
  for *j=i to N* **do**
    if $i \neq j$ **then**
      $d := |r_i - r_j|$;
      $Q[i] := q_i * q_j/d$;
    end
  end
end

Algorithm 1: The basic approach to the N body problem

The advantages to the $O(n^2)$ approach are it's simplicity, ease of implementation, and it's low

overhead. However, it's primary disadvantage is that it is limited to small numbers of particles by it's $O(n^2)$ complexity. These advantages and disadvantages are further discussed in Chapter 3

2.2 The particle mesh ewald method

2.2.1 Background

Potential vs. Energy

Energy and potential are related but different concepts. They both describe effects charged particles have on their surrounding environment, however, *Energy* is a property of a particle, and *Potential* is a property of a field. *Energy* is defined in terms of a particle-particle interaction, while *Potential* describes a field from one charged particle.
Coulomb's law describes the potential from at r_2 from a charge q_1 at r_1 as

$$V = \frac{1}{4\pi\epsilon_0}\frac{q_1}{|r_2 - r_1|}$$

And energy from a particle with charge q_2 at r_2 as $E = q_2 V$.
For our purposes, the constant term $\frac{1}{4\pi\epsilon_0}$ is material dependent, and is discarded.

Ewald summation

The key concept behind the Particle Mesh Ewald method is that of *Ewald Summation*. Ewald Summation splits the energy between two particles into two components, the long range force and the short range force. [8]

$$\phi(r) = \phi_{\text{sr}}(r) + \phi_{\text{lr}}(r)$$

The advantage of doing this is that ϕ_{sr}, the short range term, can be converges quickly in real space, while ϕ_{lr} converges quickly in reciprocal space. The Particle Mesh Ewald method takes advantage of this by calculating the ϕ_{sr} term by calculating potential directly for nearby particles, and uses a grid based direct fourier transformation to calculate the long range component.

Real space computation

The short range potential at a point can be computed by considering only particles within a small radius of the point. We can keep the radius small as ϕ_{sr} converges quickly in real space. This is also important, as we need to keep the number of particles considered less than N, in order to reduce the algorithm from $O(n^2)$ complexity. This is the case though, as the radius we consider is constant and less than the size of the simulation cell width, and we assume the particles are distributed randomly within the simulation cell.

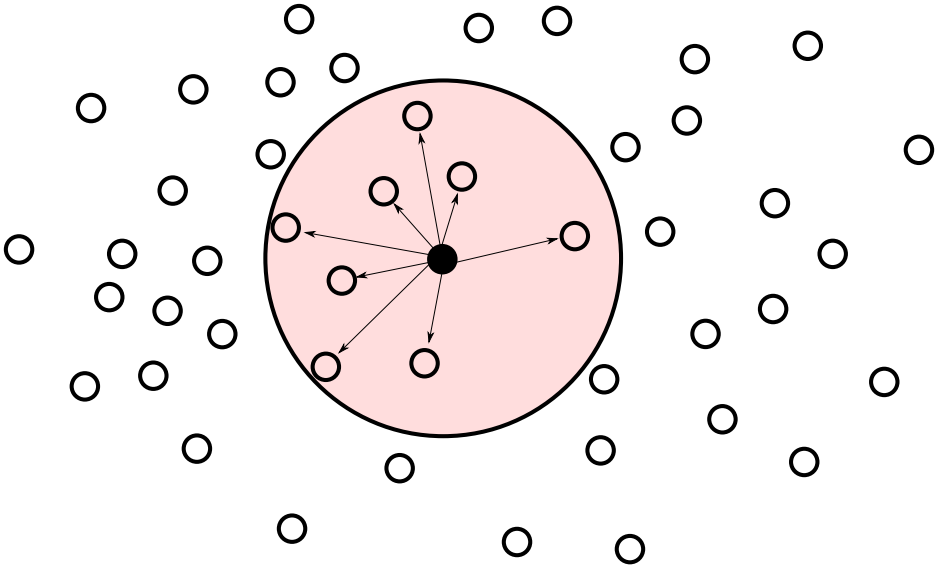

Figure 2.2: A particle and the particles that we consider it's interactions with, based on the cutoff distance

Reciprocal space computation

The reciprocal space is the long range part of the potential computation, and which converges slowly in real space. However, in reciprocal space, it converges quickly, so we use discrete fourier transformations to calculate this part of the sum.

Discrete fourier transformations require a discrete space to transform, however, our real space is continuous. So for this, we need to discretise the charges onto a grid. The approach taken in the original paper was to use Lagrangian interpolation to achieve this. Close mesh cells receive most of the charge from each particle, and this amount decreases to zero at some point, depending on the interpolation order. This is described in detail in Section 2.2.2.

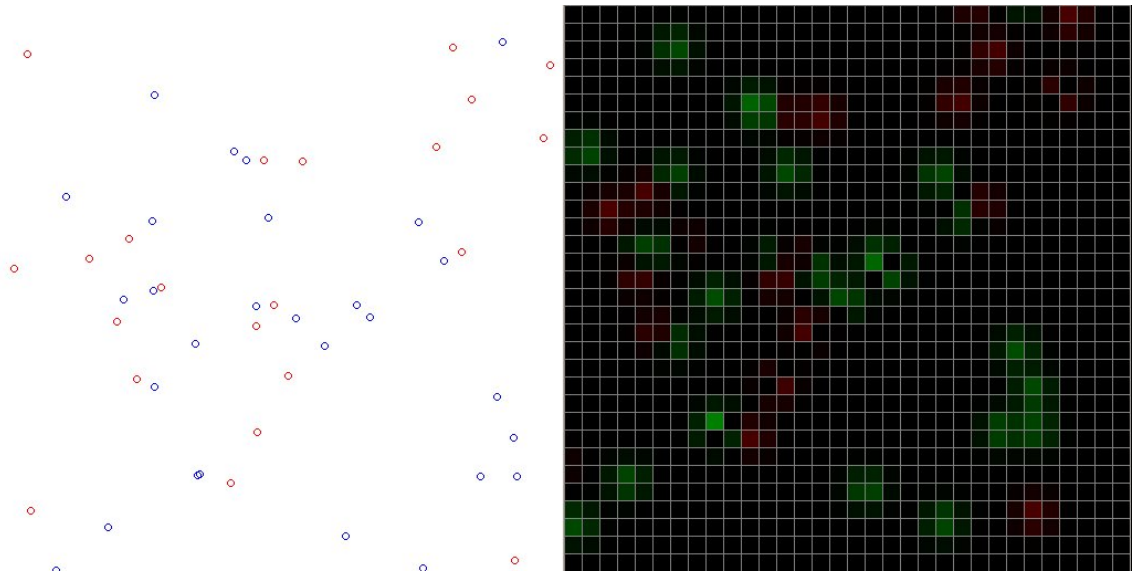

Figure 2.3: A distribution of continuous charges, and an interpretation of them as discrete charges

An alternative to using lagrange interpolation is to use cardinal B splines. This modification is known as the *Smooth Particle Mesh Ewald* method, and is advantageous in terms of accuracy, and is also easily differentiable, which is important if the forces are required as well as the potentials. [3] This is the method that was implemented in this paper.

Periodic boundary conditions

One advantage that the Particle Mesh Ewald method has is the ability to simulate *Periodic Boundary Conditions* [3]. Periodic Boundary Conditions have a wrap around behaviour - that is, instead of treating the walls of the simulation unit as empty space, we can allow them to loop around to the other side of the box again, effectively allowing infinite replication of the simulation unit. This is helpful practically, as many applications are interested in the effects on a small portion of a large system [1] (For example, simulation of water at a molecular scale). Simulating the entire system would be time consuming if not impossible, and simulating only a small portion without Periodic Boundary Conditions would produce unrealistic results (Molecules may disperse into empty space over time). While useful practically, Periodic Boundary Conditions do not change the complexity or performance, and as our interest is in a comparison with the Fast Multipole Algorithm, we do not implement Periodic Boundary Conditions in this paper.

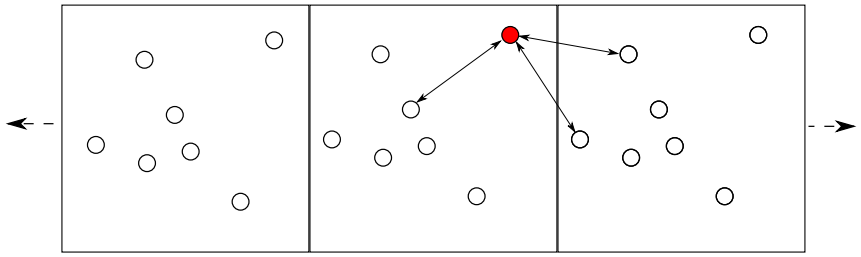

Figure 2.4: A particle (in red) interacting with it's nearest neighbours, with periodic boundary conditions in one dimension (x axis)

2.2.2 Mathematical description

Interpolating the charges to the Q array

We first present the formulation of the Q array based on lagrangian interpolation.

$$Q(k_x, k_y) = \sum_{i=1}^{N} q_i W_{2p}(u_{xi} - k_x) * W_{2p}(u_{yi} - k_y) \tag{2.1}$$

Where N is the number of particles,
n_1, n_2 are integers $< N$,
W_{2p} is a Lagrangian polynomial of order p with a value in the range $[0, 1]$,
K is the number of cells we split the grid into,
u_{xi}, u_{yi} are scaled fractional coordinates of particle i (Scaled fractional coordinate coordinates meaning $u_{xi} = K * (r_{xi}/(\text{simulation width}))$, r_{xi} is the particle i's x coordiante, so $0 \leq u_{xi} \leq K$). More information can be found in [3]

We can replace W_{2p} in the above with M_n, a cardinal B spline of order n, and the formulation of the Q array is the same. This modification is known as the Smooth Particle Mesh Ewald method. From this point forward, we will use cardinal B spline interpolation.

Calculating electrostatic potential from the Q array

With this Q array we can calculate the long range contribution to the electrostatic potential in reciprocal space.

The reciprocal space contribution to the electrostatic energy can be written as,

$$E_{\text{rec}} = \frac{1}{2 * \pi * V} \sum_{m \neq 0} \frac{\exp(-\pi^2 m^2/\beta^2)}{m^2} B(m_1, m_2) S(m) S(-m) \tag{2.2}$$

Where V is the volume (or in the two dimensional case, area) of the simulation cell,
β is the ewald coefficient,
S is the structure factor.
B is the matrix of B spline inverse fourier transform moduli, $B(m_1, m_2) = |b_1(m_1)|^2 * |b_2(m_2)|^2$. More detail on this can be found in [3].
It is shown in [3] that $S(m) \approx F(Q(m))$, so we can rewrite this as a convolution

$$E_{\text{rec}} = \frac{1}{2} \sum_{m_1=0}^{K} \sum_{m_2=0}^{K} Q(m_1, m_2) * (\theta_{\text{rec}} \star Q)(m_1, m_2) \tag{2.3}$$

With $\theta_{\text{rec}} = F(B * C)$, and so $(\theta_{\text{rec}} \star Q)(m_1, m_2) = F(B * C * F^{-1}(Q))$ [3] [7]
Where C is the matrix for the original exponential term from Equation 2.2, that is,

$$C(m_1, m_2) = \frac{1}{\pi V} \frac{\exp(-\pi^2 m^2/\beta^2)}{m^2} \text{ for } m \neq 0, C(0,0) = 0$$

Interpolating energies back from the grid in real space

If we have a point r in real space that we wish to calculate the potential for, we can interpolate back from a mesh in the same way we did while creating the discrete mesh. That is, with $r = (r_x, r_y)$, which has scaled fractional coordinates (u_x, u_y) (see Section 2.2.2), at point r we have

$$E(r_x, r_y) = \sum_{i,j=-n}^{n} Q(\lfloor u_{x+i} \rfloor, \lfloor u_{y+j} \rfloor) * M_n(u_x - \lfloor u_{x+i} \rfloor) * M_n(u_y - \lfloor u_{y+j} \rfloor) \tag{2.4}$$

The ewald coefficient

The ewald coefficient, β, is a number describing the ratio between the real space and the reciprocal space contributions to the calculation of the total energy. In practice, it depends on the tolerance ϵ_{tol}, and our desired cutoff distance r_{cut} in the following way, [2] [3]

$$\frac{\text{erfc}(\beta r_{\text{cut}})}{r_{\text{cut}}} \leq \epsilon_{\text{tol}} \tag{2.5}$$

Where erfc is the complimentary error function, a function that tends quickly to zero, depending on it's argument. The relation means that for $r > r_{cut}$, we have $\frac{\text{erfc}(\beta r_{\text{cut}})}{r_{\text{cut}}} \leq \epsilon_{\text{tol}}$, and for all $r > r_{\text{cut}}$ we can ignore the direct energy contribution, given by equation 2.6 [3]

$$E_{\text{dir}} = \frac{1}{2} \sum_{n} \sum_{i,j=1}^{N} \frac{q_i q_j \text{erfc} \beta |r_j - r_i|}{|r_j - r_i|} \tag{2.6}$$

Which becomes approximately

$$E_{\text{dir}} \approx \frac{1}{2} \sum_{n} \sum_{i,j=1}^{*N} \frac{q_i q_j}{|r_j - r_i|} \tag{2.7}$$

Where the $*$ indicates terms with $|r_j - r_i| > r_{\text{cut}}$ are left out of the sum

2.2.3 The algorithm

Main particle mesh ewald flow

The basic flow of the algorithm is given below

Data: Q: Charge assignment matrix, r_{cut}: Cutoff distance, r: position at which we calculate the potential
Initialise the ewald coefficient (Equation 2.5)
Calculate the B spline coefficients (Details in [3] [7])
Allocate particles to their cells (Section 2.2.3)
Initialise the Q matrix (Equation 2.1)
Calculate reciprocal energy (Equation 2.3)
Calculate direct energy (Equation 2.6)
Interpolate reciprocal energies back to desired coordinates (Section 2.2.2)
for *Every particle p within* r_{cut} *of r (Calculate using Verlet list, (Section 2.2.3)* **do**
  Caclulate and sum the direct potential from p at r.
end
Combine the interpolated energy and directly computed energy

Verlet list algorithm

The following algorithm utilises a list known as a *Verlet list* to keep track of which particles are within a cutoff distance of each other, with only $O(n)$ complexity. It is used in the direct energy calculation of the Particle Mesh Ewald method

for *i=0 to N* **do**
  $\text{cell}_x := \lceil u_{xi} \rceil$; (With u_{xi} the scaled fractional x coordinate, as defined in Equation 2.1))
  $\text{cell}_y := \lceil u_{yi} \rceil$; (With u_{yi} the scaled fractional y coordinate))
  direct range := $\lceil r_{cut}/\text{mesh cell width} \rceil$
  for $\delta_x = -$*direct range to* $+$*direct range* **do**
    for $\delta_y = -$*direct range to* $+$*direct range* **do**
      if $cell_x + \delta_x$ *and* $cell_y + \delta_y$ *are in the mesh* **then**
        Add i to the array closeParticles[$\text{cell}_x + \delta_x$][$\text{cell}_y + \delta_y$]
      end
    end
  end
end

After this, the array closeParticles[x][y] contains all particles within r_cut distance from a particle contained in mesh cell x, y.

2.2.4 The implementation

The Particle Mesh Ewald method method was implemented in Java, with a GUI front end. In this section, we break down the Particle Mesh Ewald method, and look at the interesting data structures and methods used in initialisation and potential calculations.

The Cardinal B Spline

The Cardinal B Spline was implemented by means of recursion. The class `BSpline.java` contains the methods required for evaluating the B Spline and it's derivative, as well as the Euler Exponential splines (the B matrix in Eq. 2.2). This recursion is performed many times, and is costly at high accuracies (high B Spline orders). This makes it a target for optimisation (See section 2.2.5)

Calculating the ewald coefficient

The ewald coefficient is calculated by means of a binary search for a coefficient that satisfies the inequality in Eq. 2.5 as closely as it can according to ϵ_{tol}. So we look for the β that minimizes

$$\left| \frac{\text{erfc}(\beta * r_{\text{cut}})}{r_{\text{cut}}} - \epsilon_{\text{tol}} \right|$$

In a binary search

Fast Fourier Transformations

[10] The chosen library for Fast Fourier Transformations (Required for calculation of the reciprocal space potential) was the JTransforms library. [10]. This library has the advantage of being implemented in pure Java code, allowing simpler integration and debugging. The Fourier Transformation operation is the core of the run time for the Particle Mesh Ewald method, and so and optimised library is essential for performance reasons.

2.2.5 Optimisations

2.3 The Fast multipole algorithm

2.3.1 Background

The Fast Multipole Algorithm is similar to the Particle Mesh Ewald method in that both use a grid structure, and bounded approximations, to speed up computation. However, they are different in the ways they use the mesh, and the approximations that are made.

Complex plane

It should be noted that for this algorithm, it is simplest to implement in two dimensions. It is possible to to implement in three dimensions, with Spherical Harmonics [5], however this is not discussed in this paper. Instead, we work in two dimensions, on the complex plane. We give a point (x, y) on the Real plane the value $z = x + yi$ on the Complex plane. This way, we can represent each point as a single number, and use complex versions of functions while calculating the multipole expansions.

Multipole expansions

A multipole expansion is a function which is a sum of a series of terms, which converges to some other function (in this case, the potential energy function). This convergence is fast, which makes it a good approximation to use in the Fast Multipole Algorithm. These multipole expansions are centered on one point, and are valid for points within a certain distance of this center, but may be shifted and combined to gain more general multipole expansions. [4]

The mesh

The mesh is similar to the one used in the Particle Mesh Ewald method. We say that the mesh has n levels, and at each level we split each cell into quarters, starting with level 0, which is the simulation cell. Each cell is split in four when moving down a level. We call the four sub cells of a cell c the *children* of c. Conversely, we call the cell that c is a sub cell of the *parent* of c.

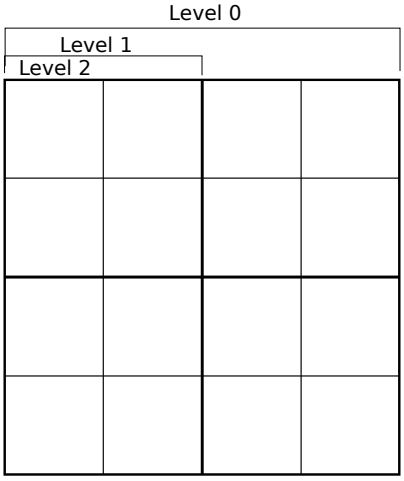

Figure 2.5: A simulation cell, with meshes up to level 2 displayed, giving $2^n = 2^2 = 4$ boxes per side

Well separated cells

For a cell c, we call a cell d *Well separated* from c if
1) Parent(d) is adjacent to Parent(c) (adjacent horizontally, vertically or diagonally)
2) d is not adjacet to c

These *well separated* cells are the ones which we will use the expansions to calculate potential for. For cells that are not well separated, we will calculate interactions directly.

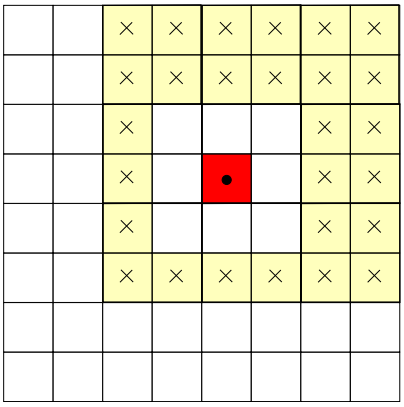

Figure 2.6: A cell, marked with a circle, and the squares that are well separated from it, marked with crosses.

2.3.2 Mathematical description

Potential and the multipole expansion approximation

We describe the potential at a point $x_0 \in \mathbb{C}$ from the charge q at $x \in \mathbb{C}$ by

$$\phi_{x_0} = -q\log|x - x_0| \tag{2.8}$$

A multipole expansion for this function is derived in [4]. Suppose that m charges of strength $\{q_i, i = 1, ..., m\}$ are located at points $\{z_i, i = 1, ..., m\}$, with $|z_i| < r$. Then for any $z \in \mathbb{C}$ with

$|z| > r$, the potential $\phi(z)$ is given by

$$\phi(z) = Q\log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k} \tag{2.9}$$

Where

$$Q = \sum_{i=1}^{m} q_i \quad \text{and} \quad a_k = \sum_{i=1}^{m} \frac{-q_i z_i^k}{k} \tag{2.10}$$

While this is exact for an infinite sum of terms, we can truncate the series at term p, and if p is large enough, this approximation is close to the actual potential $\phi(z)$. A description of how close can be found in [4]

Shifting multipole expansions

A multipole expansion's center may be shifted, which is necessary for the Fast Multipole Algorithm. Suppose that

$$\phi(z) = a_0\log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k} \tag{2.11}$$

Describes a multipole expansion which is centered on z_0. We can then shift this multipole expansion to be centered at the origin,

$$\phi(z) = a_0\log(z) + \sum_{l=1}^{\infty} \frac{b_l}{z^l} \tag{2.12}$$

Where

$$b_l = \left(\sum_{k=1}^{l} a_k z_0^{l-k} \binom{l-1}{k-1} - \frac{a_0 z_0^l}{l} \right) \tag{2.13}$$

Note that this procedure of shifting to the origin is equivalent to shifting from any point a to point b, if we treat point b as the origin, and $a - b$ as our previous multipole expansion center.

Local expansions

We can find a local expansion about the origin due to a set of charges within radius R of z_0, with $|z_0| > (c+1)R, \quad c > 1$. This local expansion is based on the multipole expansion at the same point.

$$\phi(z) = \sum_{l=0}^{\infty} b_l * z^l \tag{2.14}$$

Where

$$b_0 = \sum_{k=1}^{\infty} \frac{a^k}{z_0^k}(-1)^k + a_0 \log(-z_0) \tag{2.15}$$

2.3.3 The algorithm

The basic flow of the algorithm is given below

Data: level-count: the number of mesh levels we create, P: the set of particles, r: a point at which we wish to calculate the potential
for *i=0 to level-count* **do**
  Initialise mesh[i] by adding all particles $p \in P$ to the appropriate cells
end
for *Each cell c in mesh[level-count]* **do**
  Form a multipole expansion at at c, using Equation 2.9
end
for *i=(levelcount-1) down to 0* **do**
  for *Each cell c in mesh[i]* **do**
    Shift each multipole expansion for the child cells of c (in mesh[i+1]) to c (Eq. 2.12);
    Combine and save these multipole expansions by addition;
  end
end
for *Each cell c in mesh[0]* **do**
  Form a local expansion at c based on c's multipole expansion;
  Shift c's local expansion to the children of c;
end
for *i=1 to level-count* **do**
  for *Each cell c in mesh[i]* **do**
    Shift c's local expansion to the children of c;
  end
end
Set the cumulative potential to 0;
for *Each cell c in mesh[level-count]* **do**
  if *c is Well Separated from d's cell* **then**
    Calculate the potential at r from c's local expansion;
    Add this potential to the cumulative potential;
  else
    Calculate the potential at r from c's particles directly;
    Add this potential to the cumulative potential;
  end
end
Return the cumulative potential;

2.3.4 The implementation

A more object oriented approach was taken in the implementation of the Fast Multipole Algorithm than in the Particle Mesh Ewald method. The primary classes the mathematics are `LocalExpansion.java` and `MultipoleExpansion.java`, which contain methods for creating, shifting, evaluating and storing the expansions described in Section 2.3.2.

An object for a mesh is implemented in `Mesh.java`, which contains a 2D array of cells, defined in `Cell.java`. The `Mesh` class contains a function called `makeCoarserMesh()` which creates a coarser mesh, shifting and merging multipole expansion in the process. This coarser mesh is saved into an array of each mesh level, from the maximum depth, to zero.

Chapter 3

Comparison of the algorithms

Chapter 4

Discussion

Bibliography

[1] John A. Board Jr. Abdulnour Y. Toukmaji. Ewald summation techniques in perspective: a survey. *Computer Physics Communications*, 95:73–92, 1996.

[2] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh ewald: An n [center-dot] log(n) method for ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092, 1993.

[3] Ulrich Essmann, Lalith Perera, Max L. Berkowitz, Tom Darden, Hsing Lee, and Lee G. Pedersen. A smooth particle mesh ewald method. *The Journal of Chemical Physics*, 103(19):8577–8593, 1995.

[4] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 1987.

[5] Alexander T. Ihler. An overview of fast multipole methods, 2004.

[6] Max Planck institute of Astrophysics. The millennium simulation project. `http://www.mpa-garching.mpg.de/galform/virgo/millennium/`. Accessed: 10/10/2012.

[7] Sam Lee. An fpga implementation of the smooth particle mesh ewald reciprocal sum compute engine (rsce). 2005.

[8] Henrik G. Petersen. Accuracy and efficiency of the particle mesh ewald method. *The Journal of Chemical Physics*, 103(9):3668–3679, 1995.

[9] Theoretical and Computational Biophysics group. Namd scalable molecular dynamics. `http://www.ks.uiuc.edu/Research/namd/`. Accessed: 10/10/2012.

[10] Piotr Wendykier. Jtransforms fft library, 2011. `http://sourceforge.net/projects/jtransforms`.