# *The Fast Multipole Method - Alternative Gradient Algorithm and Parallelization*

*Ivo Kabadshow*

**Berichte des Forschungszentrums Jülich ; 4215**
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-4215

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

# Aufgabenstellung

Im Rahmen dieser Diplomarbeit soll das im HPC-Chem Projekt entwickelte FMM-Programm um die Berechnung des Gradienten erweitert werden. Durch die damit mögliche Berechnung der Kräfte kann das FMM-Programm als eigenständiges Modul in Molekulardynamikrechnungen eingesetzt werden. In diesem Zusammenhang sind die mathematischen Grundlagen des FMM-Gradienten zu erarbeiten. Es sind insbesondere Grenzwerte unbestimmter Ausdrücke auszuwerten, die sich aus den Ableitungen der Multipolmomente ergeben. Auf der Grundlage des Programmpakets *Global Arrays* soll eine massiv parallele Version des FMM-Programms entwickelt werden. Globale Datenverarbeitung und einseitige Kommunikation sind auf die Struktur der FMM abzubilden.

# Acknowledgements

I would like to express my gratitude to the following people for there support and assistance in developing this diploma thesis.

I am deeply indebted to my supervisor Dr. Dachsel whose suggestions and encouragement helped me in all the time of research. I am also much obliged for the encouragement from Prof. Radehaus who supported me throughout the last five years of my studies in Chemnitz and made it possible to write this thesis in cooperation with the Research Centre Juelich. Special thanks go to my colleague Michael Hofmann for numerous discussions and hints on parallelization.

I am grateful to all those who helped solving the occurring small and large problems, especially all people at the institute of applied mathematics at Research Centre Juelich.

## Abstract

This thesis describes the Fast Multipole Method (FMM). The method reduces the complexity of the Coulomb problem from $O(N^2)$ to $O(N)$ and is therefore called a fast Coulomb solver. The FMM is advantageous for the calculation of pairwise interactions, especially for large systems. This work is divided in three parts. The first part addresses the fundamentals of the FMM. The second part discusses the force calculation with the gradient. Two different implementations of the gradient are discussed. The last part shows the parallelization of the FMM. The procedure is described exemplarily for one pass.

## Zusammenfassung

Diese Diplomarbeit befasst sich mit der schnellen Multipolmethode (engl. FMM). Die Komplexität des Coulomb Problems lässt sich mit Hilfe der FMM von $O(N^2)$ auf $O(N)$ reduzieren. Die FMM zählt damit zu den schnellen Coulomb Lösern. Für die Berechnung der Coulomb Energie von großen Teilchensystemen besitzt die Multipolmethode durch ihre lineare Komplexität Vorteile gegenüber der direkten Berechnung. Die Arbeit behandelt drei Gebiete. Im ersten Teil werden die Grundlagen der FMM erläutert. Im zweiten Teil wird die Berechnung des FMM Gradienten vorgestellt. Dabei werden zwei Möglichkeiten der Berechnung aufgezeigt. Der letzte Teil der Arbeit befasst sich mit der Parallelisierung der FMM. Dabei wird das Vorgehen exemplarisch an einem Teilschritt der FMM erläutert.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

There are two ways to understand microscopic coherences in our macroscopic world. Firstly, it is possible to carry out experiments by using adequate equipment. Secondly, we could use a theoretical model and a computer simulation to obtain the microscopic properties of a system. Such simulations, especially particle simulations, have a broad range of application, namely in chemistry, biophysics or astrophysics. Computations on the molecular level can provide a picture of the molecular structure or dynamics. Thereby, computer simulations bridge scientific theories to practical experiments. Theories can be tested against the result from experiments. Another goal is to obtain results of molecular properties without running expensive experiments.

A central problem in computational physics is the simulation of particle systems. When the interaction between these particles is described using an electrostatic or gravitational potential, the accurate solution poses several problems. Firstly, for a straightforward computation of all mutual interactions of $N$ particles, $\frac{1}{2}N(N-1)$ pairs need to be considered [37]. This strategy has a complexity of $O(N^2)$.

Secondly, simulation usually runs over several thousand time steps. That implies the $N$-body system has to be solved over and over again. Obviously, the limiting part comes from the pairwise interaction that bounds the simulation to only small system size. This fact complicates the calculation of large and more realistic systems demanded by many scientific applications [5]. Even today's computer power is not sufficient to tackle problems with sizes larger than hundreds of thousands of particles.

To avoid the computation of all pairwise interactions and reduce the overall complexity of order $O(N^2)$ many solutions have been proposed. A first approach could be a simple truncation of the infinite potential to a feasible level. The computation time decreases and complexity reduces to $O(N)$ [7], however the calculation suffers from truncation errors [24]. Thus, a simple truncation method is ineligible when the error must be bound to a certain value.

Another approach is building special purpose hardware, such as the Grape project [12] (Gravity pipe). Rapid evaluation is achieved by parallel computation on more than 1000

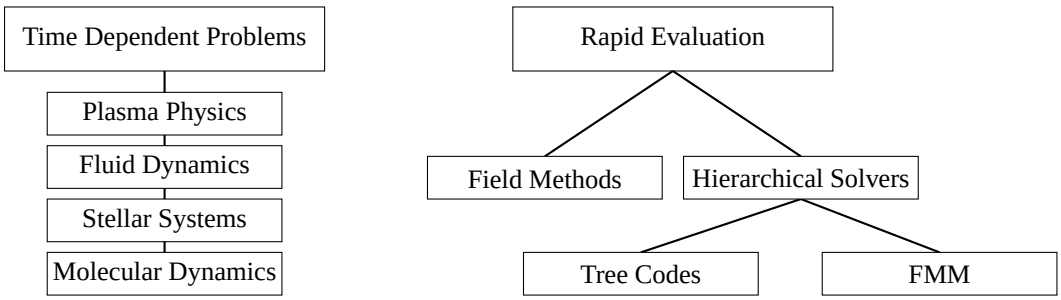
| Year | Method | Complexity | Reference |
|---|---|---|---|
| 1820 | direct summation | $O(N^2)$ | Laplace |
| 1921 | Ewald summation | $O(N^{3/2})$ | Ewald [15] |
| 1977 | Multigrid summation | $O(N)$ | Brandt [6] |
| 1986 | Barnes-Hut treecode | $O(N \log N)$ | Barnes, Hut [3] |
| 1987 | Fast multipole method | $O(N)$ | Greengard, Rokhlin [19] |
| 1988 | Particle mesh | $O(N \log N)$ | Hockney, Eastwood [21] |
| 1993 | Particle mesh Ewald | $O(N \log N)$ | Darden [11] |

Table 1.1: Historical development of fast summation algorithms.

processors. Nevertheless, the algorithmic bottleneck of quadratic runtime remains.

To overcome this limitation, there is a need for special algorithms. Cutting the potential is suitable for short-range interactions only, it does not hold for long-range interactions looming in electrostatics or gravity. Historically, significant developments started about 40 years ago. With the advent of particle-mesh methods, orders of magnitude more particles could be handled easily. For evenly distributed particles and low precision, this method is efficient and accomplishes larger particle numbers. Unfortunately for clustered systems (e.g. star clusters), this method lacks performance [17]. After 1990, algorithms called "hierarchical codes" or "tree codes" were proposed. These algorithms were developed to handle such non uniform distributions of particles. The basic idea behind these methods is to group distant particles and compute only the interaction between these groups. The amount of time needed for this kind of computation reduces to $O(N \log(N))$. Finally, the Fast Multipole Method developed by Greengard and Rokhlin [19] can be seen as a closely related scheme reducing the complexity further to $O(N)$ [34].

The FMM has been called one of the ten most significant algorithms in scientific computation discovered in the 20$^{\text{th}}$ century [14].




(a) FMM applications (b) Methodical classification

Figure 1.1: (a) The numerical $N$-body problem has many applications. The computed interactions are used to obtain a force field or a velocity field (fluid dynamics). If the problem is static the simulation is used to obtain the potential energy (Coulomb energy) of the system. Figure (b) shows the historical progress in the solution of the $N$-body problem of gravity. Field methods were established in the seventies. Hierarchical solvers were developed for the first time in the eighties [17].

# 2 The Fast Multipole Method (FMM)

## 2.1 FMM Fundamentals

Firstly, consider the electrostatic case, where a set of $N$ particles interact with each other. In this case the total Coulomb energy $E$ is given by:

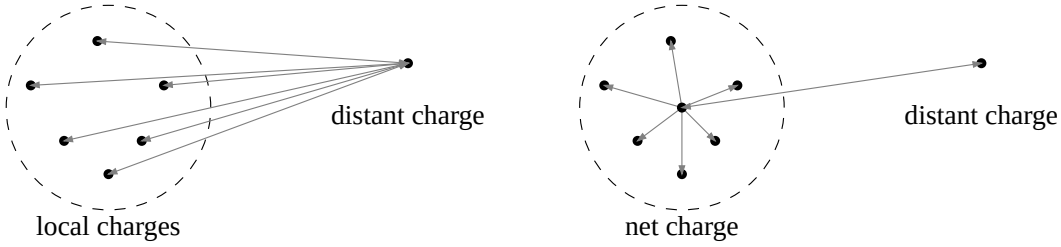
$$E = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{q_i \, q_j}{r_{ij}}. \tag{2.1}$$

The total energy is formed by taking the sum of all pairwise interactions. The charge of a particle is denoted by $q_i$, and the position of the particle is defined by $\mathbf{r}_i$. Accordingly, $r_{ij}$ defines the distance between two particles with charges $q_i$ and $q_j$. The quadratic complexity of the direct summation is obtained, because each inverse distance pair $1/r_{ij}$ contributes to the sum. A fast algorithm relies on substituting certain parts of the direct summation by a series expansion (See Figure 2.1). Instead of calculating "distant" particle interactions individually the charges are treated as a cluster, i.e. a single pseudo charge [4].

### 2.1.1 Expansion of Charge-Charge Interactions

By help of algebraic transformations, it is possible to factorize the inverse distance $1/r_{ij}$. According to [38], the system has to be transformed to spherical coordinates. Figure 2.2 illustrates this procedure for two charges. The first charge is located at a point $R$ with spherical coordinates $(r, \partial, \phi)$. The second charge is locate at point $A$ with spherical coordinates $(\alpha, \alpha, \beta)$. Thus, spherical moments $\omega_{lm}$ of a multipole expansion and coefficients $\mu_{lm}$ of a Taylor-like expansion can be defined.
The potential in three dimensions can be defined as follows:

$$\Phi(\mathbf{a}) = \frac{q}{|\mathbf{r} - \mathbf{a}|}, \tag{2.2}$$




(a) Individual interaction (b) Grouped interaction

Figure 2.1: (a) Local charges inside the disk interact individually with a distant charge. However the contribution from one particle within the disk interacting with the distant particle is almost equal to contributions from other particles in the disk concerning the distant particle. (b) All particles within the disk are approximated by a pseudo particle at disk center. Now, only the pseudo particle interacts with the distant charge.

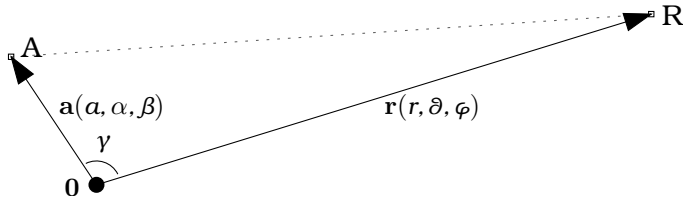



Figure 2.2: Points $A$ and $R$, with angle $\gamma$ subtended between $\overline{OA}$ and $\overline{OR}$.

where $|\mathbf{r}-\mathbf{a}|$ is the distance between a particle at point $R$ with charge $q$ and the evaluation point $A$. In terms of an infinite sum over Legendre polynomials the potential can be expressed as follows:

$$\Phi(\mathbf{a}) = \frac{q}{|\mathbf{r}-\mathbf{a}|} = q\sum_{l=0}^{\infty}\frac{a^l}{r^{l+1}}P_l(\cos\gamma), \quad (a<r). \tag{2.3}$$

Unfortunately this equation is not suitable yet, because the angle $\gamma$ depends on both coordinates. In terms of spherical harmonics the Legendre polynomial can be factorized using the Addition Theorem:

$$P_l(\cos\gamma) = \sum_{m=-l}^{l} Y_l^{-m}(\alpha,\beta)Y_l^m(\vartheta,\phi). \tag{2.4}$$

Now, the inverse distance is completely factorized. It is possible to split the equation into parts that only depend on $(a,\alpha,\beta)$ and parts only depend on $(r,\vartheta,\phi)$. This transformation is essential to reduce the complexity from $O(N^2)$ to $O(N\log N)$. The spherical harmonics can

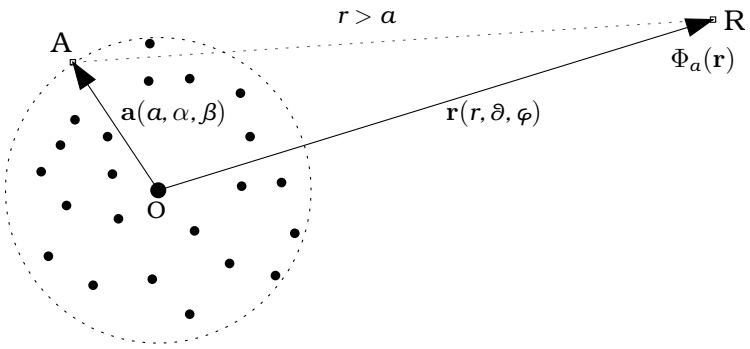



Figure 2.3: The potential at point $R$ of particles in a sphere within radius $a$ can be expressed by a multipole expansion around 0.

be expressed in terms of associated Legendre polynomials:

$$Y_l^m(\alpha,\beta) = \sqrt{\frac{(l-|m|)!}{(n+|m|)!}} P_{lm}(\cos\alpha) e^{im\beta}, \tag{2.5}$$

$$Y_l^m(\partial,\phi) = \sqrt{\frac{(l-|m|)!}{(n+|m|)!}} P_{lm}(\cos\partial) e^{im\phi}, \tag{2.6}$$

where $i^2 = -1$. Further transformations of spherical harmonics finally lead to:

$$\frac{1}{|\mathbf{r}-\mathbf{a}|} = \sum_{l=0}^{\infty} \sum_{m=-l}^{m=l} \frac{(l-m)!}{(l+m)!} \frac{a^l}{r^{l+1}} P_{lm}(\cos\alpha) P_{lm}(\cos\partial) e^{-im(\beta-\phi)}. \tag{2.7}$$

Equation (2.7) can be divided into two parts, a local part with its multipole expansion $\omega_{lm}$ (See Figure 2.3) and a distant part with its Taylor-like expansion $\mu_{lm}$. Thus, chargeless moments $O_{lm}$ of a multipole expansion are defined to be:

$$O_{lm}(a,\alpha,\beta) = a^l \frac{1}{(l+m)!} P_{lm}(\cos\alpha) e^{-im\beta}. \tag{2.8}$$

The corresponding Taylor-like expansions can be formulated as:

$$M_{lm}(r,\partial,\phi) = \frac{1}{r^{l+1}} (l-m)! P_{lm}(\cos\partial) e^{im\phi}. \tag{2.9}$$

The coefficients of a multipole expansion including charges are denoted as $\omega_{lm} = qO_{lm}$, the

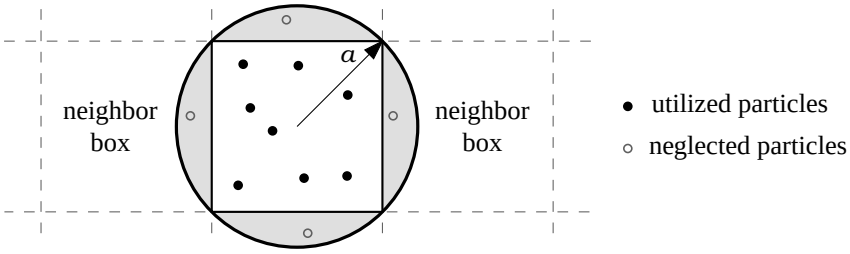



Figure 2.4: Formation of multipoles in each box. All particles within radius $a$ and within the same box are expanded into multipoles of the given box. Particles within the sphere but not in the considered box are neglected.

coefficients of a Taylor-like expansion including charges are denoted as $\mu_{lm} = qM_{lm}$. Now, the inverse distance is given by:

$$\frac{1}{|\mathbf{r}-\mathbf{a}|} = \sum_{l=0}^{\infty} \sum_{m=-l}^{m=l} O_{lm}(\mathbf{a}) M_{lm}(\mathbf{r}). \tag{2.10}$$

### 2.1.2 Pass 1: Calculation and Shifting of Multipole Moments

In order to calculate the Coulomb energy and Coulomb forces the input particle positions and particle charges are considered in a simulation box of finite size [38]. For simplicity and numerical stability this simulation box is scaled into the computation box with range $[0..1, 0..1, 0..1]$. To apply the expansion described above, the computation space has to be divided in a tree-like manner. In the first step, the computation box is divided into eight child boxes of equal size. Thus, the main box is divided half along each coordinate axis (Figure 2.13 on page 17). The child boxes extend across a subset of the entire computation box. The particles can be sorted into the generated child boxes. Then each child box is subdivided again recursively. The goal of this subdivision is to keep the particles at the lowest level independent of the total number of particles. Otherwise, the algorithm would not have a complexity of $O(N)$ [38]. On each new level, the particles are sorted into the child boxes again. The spatial partitioning proceeds until a certain number of subdivisions is obtained. Now at the lowest level the corresponding multipole expansion can be computed for each box. To minimize computational overhead, all empty boxes are pruned. Therefore, only multipole moments of non-empty boxes are stored. After setting up all multipole expansions at the lowest level, these expansions are shifted up the tree via the operator **A** (See section

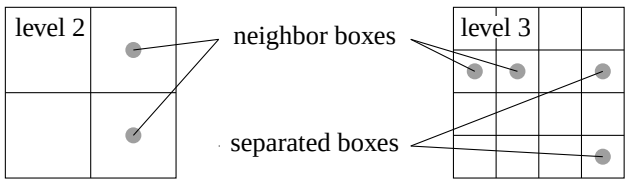
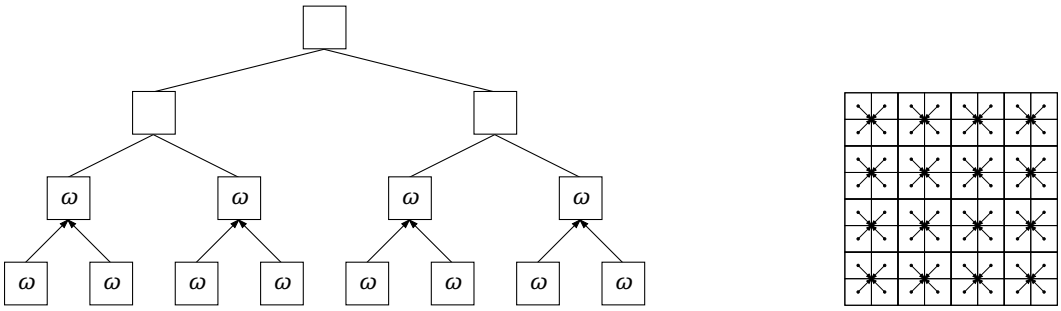



Figure 2.5: Left: Tree level 2 – All boxes shown are neighbor boxes. It is not possible to choose two boxes such that another box is in between. Right: Tree level 3 – There exist separated boxes ($ws = 1$). It is possible to place two gray dots such that at least one box is in between.




(a) Pass 1: Shifting of $\omega_{lm}$

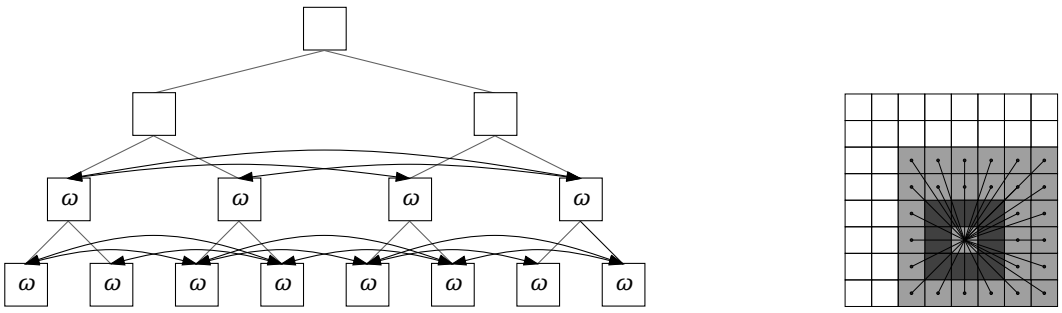
(b) Pass 1 – 2D view

Figure 2.6: Pass 1: Formation and shift of multipole moments. Arrows denote shifts from the center of a child box to the center of the parent box.

2.1.7). This procedure is repeated until the third tree-level is reached (See Figure 2.6). This is the highest level where mutually independent boxes exist.

### 2.1.3 Pass 2: Transforming Multipole Moments

As the expansion of the potential converges for $r > a$ only, neighboring boxes cannot interact via multipoles. Figure 2.4 illustrates this scheme. Gray-shaded charges are encircled by radius $a$ but do not share the same box. Therefore, a separation criterion has to be defined. The criterion $ws$ is called "well separateness" and influences the convergence of the expansion and thus the accuracy of the calculation. It can take any value from 1 to the number of boxes $n$ in one direction. For $ws = 1$, only nearest neighbor boxes are skipped, for $ws = 2$, also next-nearest neighbor boxes are skipped, and so on. For large values of $ws$, the FMM behaves like the direct summation, because no box can interact via expansions. On a given




(a) Pass 2: Transform remote $\omega_{lm}$ to local $\mu_{lm}$

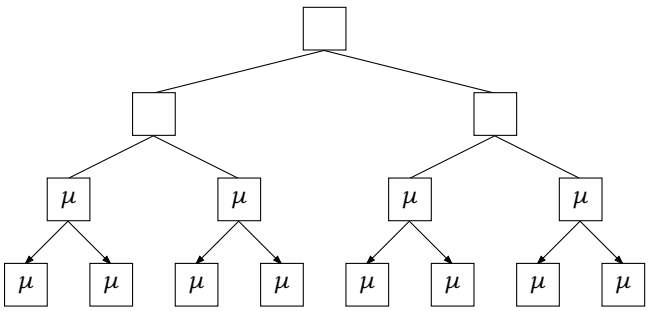
(b) Pass 2 – 2D view

Figure 2.7: Pass 2: Multipole interaction. Only well-separated boxes on each level interact.

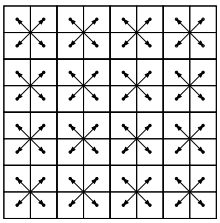
level, e.g. the lowest level, only boxes interact which are well separated but their parents are not. This fact leads to significant speedup, because more distant particles do not interact on this level. After evaluating all transformations at one level, the transformation proceeds to the next higher level until no separated boxes remain (See Figure 2.7). On level two all boxes are non-separated (See Figure 2.5) and therefore cannot interact via multipoles. For $ws = 1$ or $ws = 2$ this would be level 3. For $ws = 3$, this procedure can be performed until level 4 is reached. Each transformation converts the distant multipole expansion around the distant origin to a local Taylor-like expansion. All Taylor-like expansions around the same origin can be summed. This scheme allows only a constant number of boxes, namely boxes that meet the interaction rule, to be transformed. Operator **B** is used to perform the $\omega_{lm} \rightarrow \mu_{lm}$ transformation.

### 2.1.4 Pass 3: Shifting Taylor-like Coefficients

In pass 2 only boxes well-separated from each other interact. In pass 3 the neglected information from all boxes which did not interact, is shifted top-down from the higher levels to lower levels, beginning at level 3 ($ws = 1$ or $ws = 2$). In result, each lowest level box contains all information from all remote boxes. This shift is carried out via the operator **C**. The information is shifted from parents to children. This pass is repeatedly performed until the lowest level is reached (See Figure 2.8).

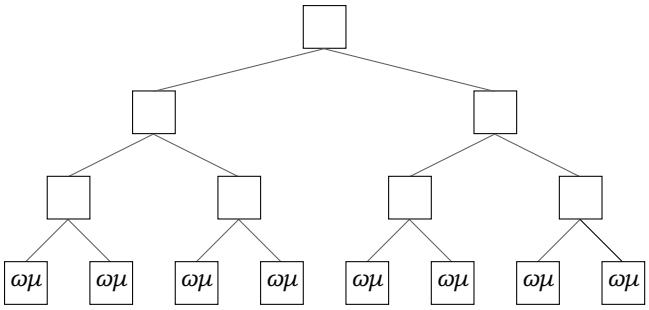



(a) Pass 3: Shifting of $\mu_{lm}$

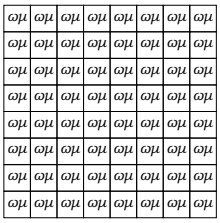


(b) Pass 3 – 2D view

Figure 2.8: Pass 3: Taylor-like expansions are shifted down the tree to complete the neglected interactions from more distant particles. Arrows denote shifts from the center of a parent box to the center of the child box.




(a) Pass 4: Lowest level boxes contain $\omega_{lm}$ and $\mu_{lm}$.




(b) Pass 4 – 2D view

Figure 2.9: Pass 4: Calculation of the far field energy in each lowest level box.

### 2.1.5 Pass 4: Calculating the Far Field Energy

After computing the previous three passes, all lowest level boxes now contain the necessary information to calculate the far field energy. Every box keeps a multipole expansion $\omega_{lm}$ of the contained charges $q_i$ and a Taylor-like expansion from all distant boxes, respectively charges (See Figure 2.9). As the multipole and Taylor-like expansion both have the same center, they can be summed up in each box. Finally summing up all the results from all boxes, the far field part of the Coulomb energy becomes:

$$E_{FF} = \sum_{ibox} \sum_{l=0}^{L} \sum_{m=-l}^{l} \omega_{lm} \mu_{lm}. \tag{2.11}$$

### 2.1.6 Pass 5: Calculating the Near Field Energy

In pass 5 all interactions, which cannot be computed via the far field approach, are calculated. All non-separated boxes around box *ibox* and the box *ibox* itself are taken into account. All particles in these boxes interact directly (See Figure 2.10). Contributions to the near field energy can be split up in intra-box and box-box interactions.

$$E_{NF_1} = \sum_{ibox} \sum_{i=1}^{N_{ibox}-1} \sum_{j=i+1}^{N_{ibox}} \frac{q_i\, q_j}{r_{ij}}, \tag{2.12}$$

Equation (2.12) shows how to calculate the interaction energy for all particles contained in box *ibox*.

$$E_{NF_2} = \sum_{ibox} \sum_{jbox} \sum_{i=1}^{N_{ibox}} \sum_{j=1}^{N_{jbox}} \frac{q_i\, q_j}{r_{ij}}, \tag{2.13}$$

Equation (2.13) shows how all pairwise interactions between box *ibox* and its neighbor boxes *jbox* defined by the separation criterion $ws$ are calculated. This pass has to be performed for all non-empty boxes at the lowest level. The far field part $E_{FF}$ is obtained in pass 4:

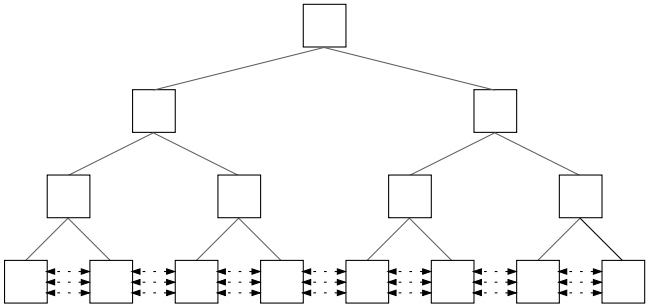
$$E_{FF} = \sum_{ibox} \sum_{l=0}^{L} \sum_{m=-l}^{l} \omega_{lm} \mu_{lm}. \tag{2.14}$$

The total Coulomb energy can be obtained by summing up all near field and far field parts:

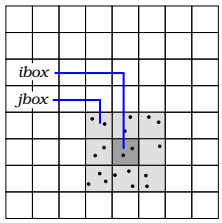
$$E = E_{NF_1} + E_{NF_2} + E_{FF}. \tag{2.15}$$

### 2.1.7 Translation Operators

To understand the computational scheme of the FMM some operators have to be defined. The algorithm needs to form multipole expansions, translate these expansions, construct local expansions from distant particles and translate those local expansions. These operators are also essential for linear scaling [38]. The use of these operators is described in Sections 2.1.2 to 2.1.5.



(a) Pass 5: Interaction on the lowest level of the FMM tree.




(b) Pass 5 – 2D view

Figure 2.10: Pass 5: Final step – all interactions not eligible for expansion are calculated directly.

### Operator A: Shifting

To shift a multipole expansion from a child box center to its parent box operator A is used. It shifts multipole expansions around $a$ to a new center around $b$. Within the algorithm, the infinite sum in Equation (2.16) is truncated to $L$ terms. Therefore a truncation error occurs. We can define the shifted multipole expansion $\omega_{lm}$ by means of the formula:

$$\omega_{lm}(\mathbf{a}+\mathbf{b}) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} A_{jk}^{lm}(\mathbf{b})\omega_{jk}(\mathbf{a}), \tag{2.16}$$

with

$$A_{jk}^{lm} = O_{l-j,m-k}. \tag{2.17}$$

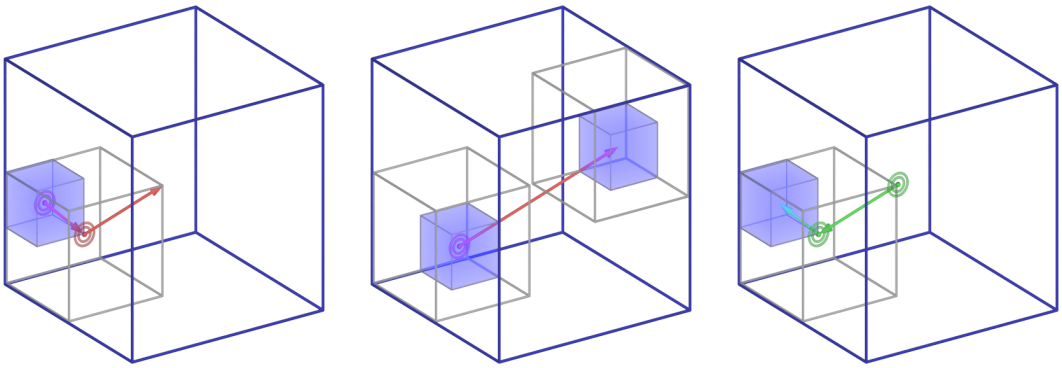
### Operator B: Transformation

To transform a local multipole expansion around **a** to a local Taylor-like expansion around $(\mathbf{b}-\mathbf{a})$ operator B is applied. Numerical calculations of operator **B** induce truncation errors as well, because only $L$ terms are considered for the sum. The transformed Taylor-like expansion can be represented as:

$$\mu_{lm}(\mathbf{b}-\mathbf{a}) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} B_{jk}^{lm}(\mathbf{b})\omega_{jk}(\mathbf{a}), \tag{2.18}$$

with

$$B_{jk}^{lm} = M_{j+l,k+m}. \tag{2.19}$$



(a) FMM Operator A (b) FMM Operator B (c) FMM Operator C

Figure 2.11: (a) Operator **A** implements translations up the tree. (b) Operator **B** implements a transformation from a multipole expansion to a local Taylor-like expansion. (c) Operator **C** implements translations down the tree.

### Operator C: Shifting

The operator **C** shifts the $L$-term local expansions from a parent box to its eight children boxes' centers. A Taylor-like expansion located in a box around $\mathbf{r}$ is shifted to its new position $(\mathbf{r}-\mathbf{b})$. On the contrary to operator **A** and operator **B**, operator **C** has no associated truncation error. Thus, the shift of local Taylor-like expansions comes without incurring additional errors. The shifted Taylor-like expansion is defined by:

$$\mu_{lm}(\mathbf{r}-\mathbf{b}) = \sum_{j=l}^{\infty} \sum_{k=-j}^{j} C_{jk}^{lm}(\mathbf{b})\mu_{jk}(\mathbf{r}), \tag{2.20}$$

with

$$C_{jk}^{lm} = O_{j-l,k-m}. \tag{2.21}$$

Omitting operator $A$ and $C$ increases complexity to $O(N \log N)$. Obviously all operators induce a complexity of $O(L^4)$. Especially high accuracy calculations are slowed down. To overcome this problem, improvements to the operators have been made. (See section 2.4).

## 2.2 Fractional Tiers

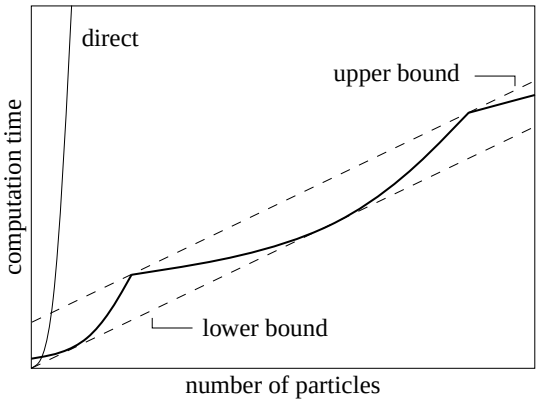
To achieve linear scaling, the number of particles at the lowest level boxes must be independent from the total number of particles. Considering a homogeneous particle system, a certain number of particles reside in each lowest level box. When adding a new charge to the system the total number of boxes should not change, since the number of particles in lowest level boxes must be independent from the total number of particles. Therefore, every added particle contributes a linear amount of computation time to the far field part and quadratical amount to the near field part. The scaling with particle number becomes locally quadratical. Introducing a new level to the FMM tree the number of boxes increase by a factor of eight. Thus, the tree depth is kept constant until the number of particles increases by a factor of eight. Figure 2.12 illustrates the fact.

To overcome the restriction that the number of boxes has to be a power of eight and achieve arbitrary numbers of lowest level boxes, the entire simulation box can be scaled. Thereby, a better balance between far field and near field calculations can be achieved. The scaling factor determines the number of particles per box. It can take values from 0.5 to 1.0. Compressing the system to half if its original size along each axis causes the tree level to increase by one. Now there are eight times more boxes, but only $1/8$ of all boxes contain particles. Varying the scaling factor between the limits 0.5 and 1.0 leads to arbitrary box numbers [39]. The locally quadratical scaling is improved. Computation time now is near to the lower bound (See Figure 2.12a).

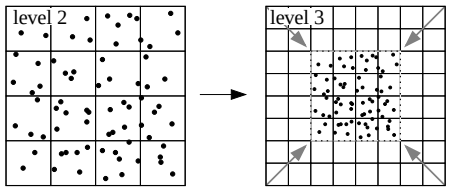
## 2.3 Error Estimations

The precision of the FMM is known *a priori*, and a tolerance parameter is defined in terms of this prescribed precision.

The FMM depends on three parameters, namely the tree depth $D$, the separation criterion $ws$ and the length of the multipole expansion $L$ (See Figure 2.13). These parameters strongly affect the accuracy and running time of the FMM. Admittedly, the user is only interested in the total Coulomb energy $E$ and the forces acting on each particle to a given precision, but not in determining optimal FMM parameters. Hence, it is impractical to leave the choice of a proper parameter set to the user. Since all parameters have much influence on both the computation time and the desirable precision, it would be preferable to have an analytic expression that identifies the optimal parameter set. Unfortunately there is no appropriate




(a) Locally quadratical scaling




(b) Compression

Figure 2.12: Locally quadratical scaling. Eight times more particles introduce a new tier. The introduction of an additional tier and the compression of the simulation box to 50% of the original size does not change the number of non-empty boxes.

method known.

However this implementation of the FMM allows the evaluation of optimal parameters automatically at runtime. The particle system could be of any kind of distribution, a homogeneous one or non-homogeneous one.

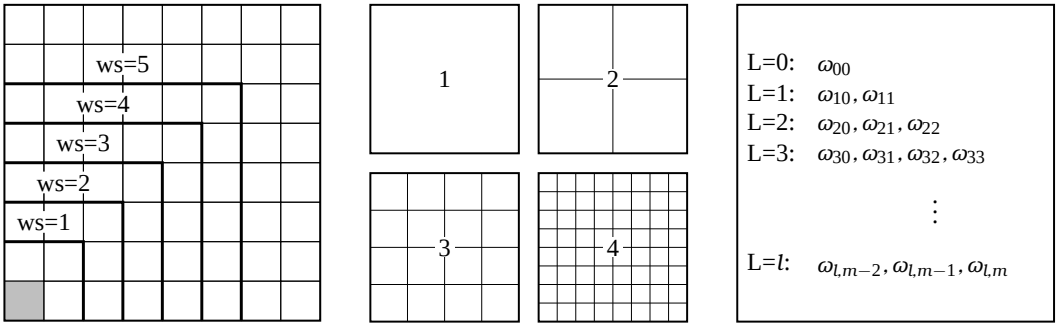
The parameters $ws$, $D$, and $L$ can be optimized such that the computation time $t$ achieves a minimum and a pre-defined error bound $\varepsilon$ is obtained, i.e. formally:

$$\frac{\partial t}{\partial ws} = 0, \qquad \frac{\partial t}{\partial D} = 0, \qquad \frac{\partial t}{\partial L} = 0, \tag{2.22}$$

and,

$$\Delta E(D, L, ws) \leq \varepsilon . \tag{2.23}$$

The separation criterion $ws$ and the multipole length $L$ can take positive integer values only. The depth of the FMM tree can also take fractional values as described in section 2.2. The determination of the optimal parameter set takes only little extra computation time in this implementation. In contrast to improper chosen parameters, which can lead to an order of magnitude higher computation time this is worthwhile. There also exist other implementations that optimize the FMM parameters [9], but interestingly all references yield different optimal parameters.




(a) Separation criterion $ws$ (b) FMM tree depth $D$ (c) Multipole length $L$

Figure 2.13: The FMM contains three parameters: (a) A separation criterion $ws$, (b) a certain tree depth $D$ and (c) a multipole expansion with length $L$.

## 2.4 Rotation-based FMM

The operators introduced in section 2.1.7 require a complexity of $O(L^4)$, with multipole length $L$ for arbitrary angles. To achieve reasonable efficiency at high precision this is a major obstacle. There are a number of schemes offering reduced complexity.

Some groups overcome this problem by shortening the length of the multipole expansion for boxes situated very far from the interacting one. Unfortunately, these approaches do not guarantee the error bound [31]. Using the full operators $O$:

$$O_{lm}(a,\alpha,\beta) = \frac{1}{(l+m)!} a^l P_{lm}(\cos\alpha) e^{-im\beta}, \tag{2.24}$$

and full operator $M$:

$$M_{lm}(r,\vartheta,\varphi) = (l-m)! \frac{1}{r^{l+1}} P_{lm}(\cos\vartheta) e^{im\varphi}. \tag{2.25}$$

it can be easily shown that a translation in the $z$ direction ($\alpha = \vartheta = 0$, $\beta = \varphi = 0$) only requires $L^3$ work. Thus, translations along the $z$-axis are less expensive. The operator $O$ degenerates to:

$$O_{lm}(a,0,0) = \frac{1}{l!} a^l \delta_{m0}. \tag{2.26}$$

The operator $M$ degenerates to:

$$M_{lm}(r,0,0) = l!\frac{1}{r^{l+1}}\delta_{m0}. \tag{2.27}$$

with $\delta_{mn}$ defined as:

$$\delta_{mn}\begin{cases} \forall\, m \neq n \rightarrow 0 \\ m = n \rightarrow 1 \end{cases} \tag{2.28}$$

To take advantage of the reduced operator costs, the following scheme has to be followed. The standard FMM could be modified such that all translations require the reduced costs:

1. Rotate the coordinate system ($L^3$ operations) such that the vector connecting the source box and the target box lies parallel to the $z$-axis.
2. Shift the expansion parallel to the $z$-axis ($L^3$ operations).
3. Rotate back to the original coordinate system ($L^3$ operations).

The problem is reduced from three dimensions to one dimension by this rotations (See Figure 2.14). Both operators $O_{lm}$ and $M_{lm}$ are necessary for the three FMM operators **A**, **B** and **C**. Using Equations (2.26) and (2.27), the operators take a simpler form. Operator A is then given by:

$$A_{jk}^{lm}(\mathbf{b}) = \frac{1}{(l-j+|m-k|)!}b^{l-j}\delta_{m-k,0}, \tag{2.29}$$

operator B takes the form:

$$B_{jk}^{lm}(\mathbf{b}) = (l+j-|m+k|)!\frac{1}{b^{j+l+1}}\delta_{m+k,0}, \tag{2.30}$$

and operator C can be formulated as:

$$C_{jk}^{lm}(\mathbf{b}) = \frac{1}{(j-l+|k-m|)!}b^{j-l}\delta_{k-m,0}. \tag{2.31}$$

The simplified translations and transformations can be written as follows. For the shift of a multipole expansion we get the formula:

$$\omega_{lm}(\mathbf{a}+\mathbf{b},q) = \sum_{j=|m|}^{l}\frac{b^{l-j}}{(l-j)!}\omega_{jm}(\mathbf{a},q). \tag{2.32}$$

The transformation of a distant multipole expansion to a local Taylor-like expansion takes

the form:

$$\mu_{lm}(\mathbf{b}-\mathbf{a},q)=\sum_{j=0}^{\infty}\frac{(j+l)!}{b^{j+l+1}}\omega_{j,-m}(\mathbf{a},q). \tag{2.33}$$

The translation of a Taylor-like expansion can then be rewritten as:

$$\mu_{lm}(\mathbf{r}-\mathbf{b},q)=\sum_{j=l}^{\infty}\frac{b^{l-j}}{(j-l)!}\mu_{jm}(\mathbf{r},q). \tag{2.34}$$

An arbitrary rotation can be split into two rotations. Firstly, the phase of the moments is changed by a rotation about the $z$-axis. Secondly, the multipole expansion is rotated about the $y$-axis. All rotations conserve the total angular momentum and can be described via Wigner matrices. It can be proven that this leads to the same results as the standard method [10]. It is clear, that especially for high accuracy calculations this implementation is more efficient.
Even further reduction is possible by using Fast Fourier Transform (FFT), reducing the operator complexity to $O(L^2\log L)$, but massive changes have to be applied and significantly more memory is required. In contrast, the rotation-based FMM does not need significant amount of additional memory. For further reading see Reference [40].

## 2.5 Crossover Point

To compare the FMM to the direct method, it is necessary to know at which particle number the method is more efficient than direct pairwise evaluation. Since the direct summation scales with $O(N^2)$ and the FMM scales with $O(N)$ there should be a particle number for which the FMM outperforms the direct summation. The management of the FMM structure, the setup of the tree and the formation of the multipole moments are not for free. So it is important to know, where the break-even point occurs. It is also clear, that this crossover point depends on the requested error. The direct summation method loses precision on every summation step due to rounding errors [32]. The FMM is also susceptible for rounding errors. However, the linear scaling will produce less rounding errors as a result of fewer floating point operations. Additionally, it is possible to set a specific error bound precisely. In Table 2.1 two crossover-points are shown. The first one with moderate relative error level of $\Delta E = 10^{-5}$ and the second one almost with machine precision[1]. The "break-even" point

[1] machine precision $\epsilon \approx 2.22 \cdot 10^{-16}$ on JUMP [22]

| $\Delta E_{rel,req}$ | $\Delta E_{rel}$ | Time[sec] | Method | Depth | Multipole Length | N |
|---|---|---|---|---|---|---|
| $10^{-5}$ | $0.77 \cdot 10^{-05}$ | 0.011 | FMM | 1.62 | 4 | 512 |
| - | $0.33 \cdot 10^{-12}$ | 0.013 | Direct | - | - | 512 |
| $10^{-11}$ | $0.22 \cdot 10^{-11}$ | 0.50 | FMM | 1.62 | 16 | 4096 |
| - | $0.97 \cdot 10^{-11}$ | 1.01 | Direct | - | - | 4096 |

Table 2.1: In this implementation of the FMM the Crossover point with moderate error of $\Delta E_{rel} = 10^{-5}$ is at 512 particles. For higher accuracy the crossover point shifts to 4096 particles. For particle numbers higher than 4096 this implementation of the FMM is always faster than direct summation, and at least as accurate as direct summation.

between the FMM and the "direct" method, i.e. the point where the two methods take the same amount of execution time, is $N \approx 512$. Note that: Even for 512 particle the direct summation loses about 3 figures of accuracy. Thus, the second crossover-point shows the FMM compared to direct summation with truncation error in the same range. All reference values were computed with quad-precision to guarantee the error bound even for direct summation.

Finding crossover points are discussed in many publications [13, 36, 25]. However, the range of the crossover points strongly depends on the implementation.

## 2.6 Test Calculations

To prove the theoretical scaling properties some test cases are considered. Two scaling properties are of particular interest. Firstly, how does the computation time increase when the system size gets larger but multipole length is fixed? Secondly, how does the computation time increase when the multipole length increases, hence the computation is more accurate, but the number of particles is fixed?

### 2.6.1 $O(N)$ Scaling with Number of Particles $N$

Increasing the system size by a factor of eight should yield a computation time eight times higher, since the FMM is a linear scaling method. Fixing the multipole length to 15, three different systems were examined. The first one contained $8^6$, the second $8^7$ and the last $8^8$ particles. The particles were evenly distributed on a grid, thus all lowest level boxes

| N | Time[hh:mm:ss][1] | Scaling | Depth | Multipole Length |
|---|---|---|---|---|
| 262144 | 0:01:09 | - | 5 | 15 |
| 2097152 | 0:09:29 | 8.25 [8] | 6 | 15 |
| 16777216 | 1:16:12 | 8.04 [8] | 7 | 15 |
| 134217728 | 10:17:57 | 8.11 [8] | 8 | 15 |

[1] All results were obtained on JUMP [22]

Table 2.2: Scaling concerning the number of particles $N$. Multipole expansion length is fixed to validate linear scaling factor of eight.

contained the same number of particles. Table 2.2 presents the obtained scaling factors. The result shows linear scaling with a factor of about eight as it was expected. The slightly higher values of 8.04, 8.11 and 8.35 are due to administration costs within the FMM. Every new level introduces a little overhead to the computation.

### 2.6.2 $O(L^3)$ Scaling with Multipole Length $L$

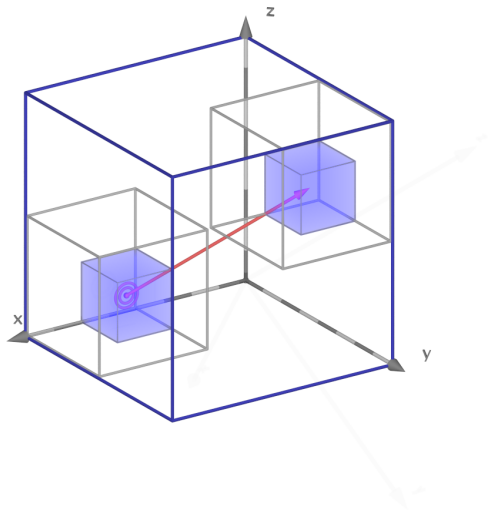
In section 2.4 all described operators showed a complexity of order $O(L^3)$. Thus, it is interesting how strong the $L^3$-term affects the computation time for realistic multipole lengths. Realistic multipole lengths are referred to computations within the numerical feasible range. Multipole lengths yielding a theoretical precision beyond machine precision are not referred as realistic. The question is: How expensive is a higher accuracy?

Table 2.3 illustrates the context. For a fixed particle system with more than one billion particles the length of the multipole expansion is increased gradually. Doubling the multipole length should slow down the computation by a factor of eight. Instead, it shows that the computation time roughly increases by a factor of two. This fact can be pointed out for almost all realistic multipole lengths. Only for very high multipole lengths yielding in results near machine precision the slope rises and shows the expected non-linear behavior. This is advantageous for high precision calculations. Note that the same calculation performed by direct summation [32] would only allow a relative error bound of about $10^{-5}$. Ten figures are simply lost due to truncation errors. To obtain this result with todays computer power this calculation would need more than 5 years on a single CPU.

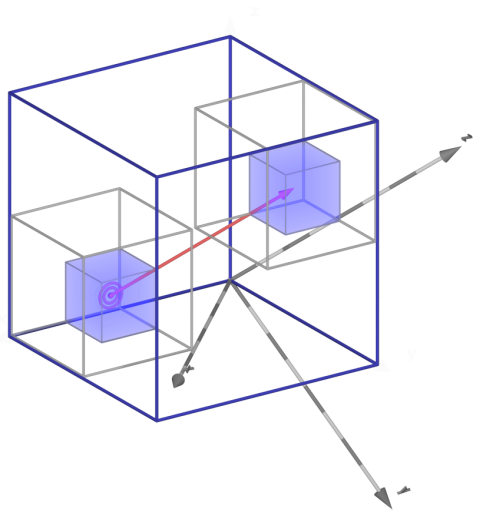
| N* | $\Delta E_{rel,req}$ | $\Delta E_{rel}$ | Time[hh:mm] | Depth | Multipole Length |
|---|---|---|---|---|---|
| 1073741824 | $10^{-03}$ | $0.2 \cdot 10^{-03}$ | 05:44 | 8.7 | 3 |
| 1073741824 | $10^{-06}$ | $0.4 \cdot 10^{-06}$ | 08:22 | 8.5 | 6 |
| 1073741824 | $10^{-09}$ | $0.4 \cdot 10^{-09}$ | 12:35 | 8.1 | 10 |
| 1073741824 | $10^{-12}$ | $0.3 \cdot 10^{-12}$ | 18:47 | 7.8 | 15 |
| 1073741824 | $10^{-14}$ | $0.7 \cdot 10^{-14}$ | 25:59 | 7.7 | 18 |

* The computation time obtained for the $8^{10}$ particles test case with $L = 15$ case cannot be compared with results from Table 2.2. This is only possible for optimal FMM parameter sets. All results from Table 2.2 have a fixed multipole length and no fractional FMM tree depths. Thus, the parameter set is not optimal.

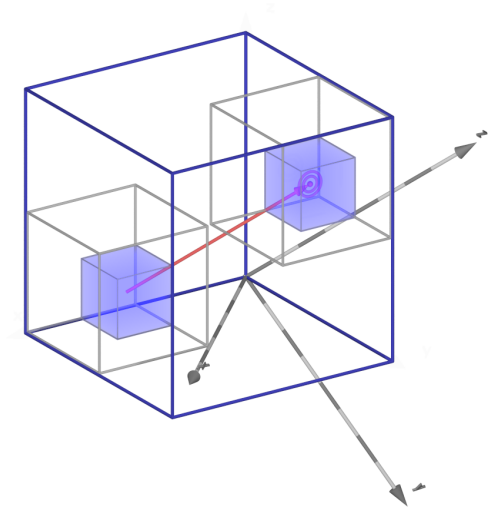
Table 2.3: Scaling concerning the length $L$ of the multipole expansion. The number of particles is fixed to $8^{10}$.



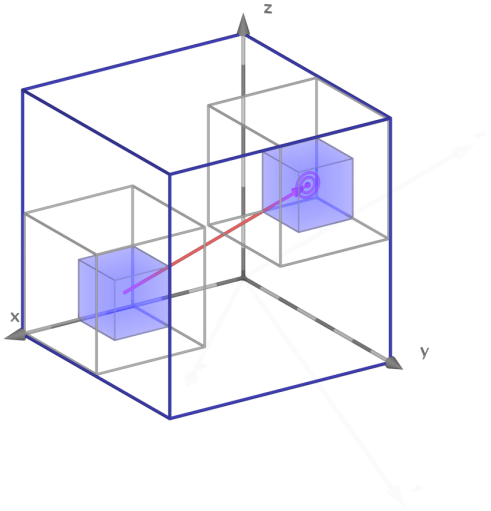

(a) The translation axis is not along the $z$-axis.



(b) Step 1: The coordinate system is rotated such that the translation axis is along the $z$-axis.



(c) Step 2: The multipole expansion $\omega_{lm}$ is transformed to local Taylor-like expansion $\mu_{lm}$.




(d) Step 3: Rotate back the coordinate system.

Figure 2.14: Rotation-based FMM.

# 3 FMM Gradient

Molecular dynamic calculations do not only require the total Coulomb energy. To describe particle systems and introduce dynamics, the forces acting on each particle are more interesting. The goal of *N*-body problems is to determine the motion over time of particles due to forces induced by other particles. The basic method used to describe the motion is to iterate using discrete time steps and repeat the following operations:

- Update positions $\mathbf{x}_i$ using velocities $\mathbf{v}_i$, $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t \mathbf{v}_i$
- Calculate forces $\mathbf{F}$
- Update velocities $\mathbf{v}_i$

In the FMM scheme, it is very easy to obtain the Coulomb forces within little extra computational effort. In general, the potential $\phi(\mathbf{r})$ is a continuous function on a coordinate space. If it is known for every value of $r$, we can calculate the forces on all particles by computing the gradient $\nabla\phi(\mathbf{r})$.

This chapter describes two ways of obtaining the forces on the system's particles. The first one is straightforward. The second one needs shift operations but does not require the evaluation of complicated derivaties of multipole moments.

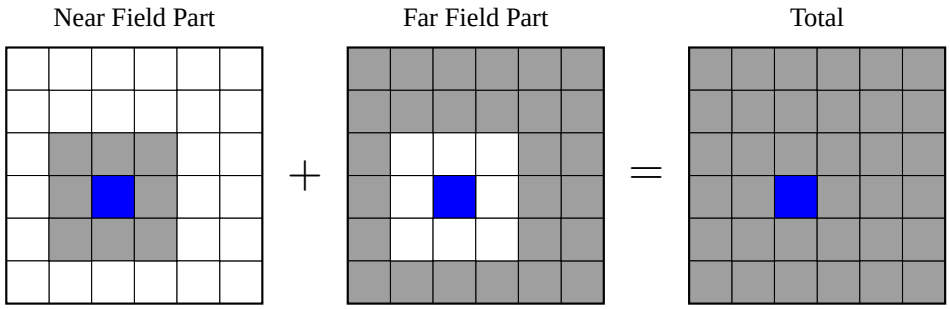
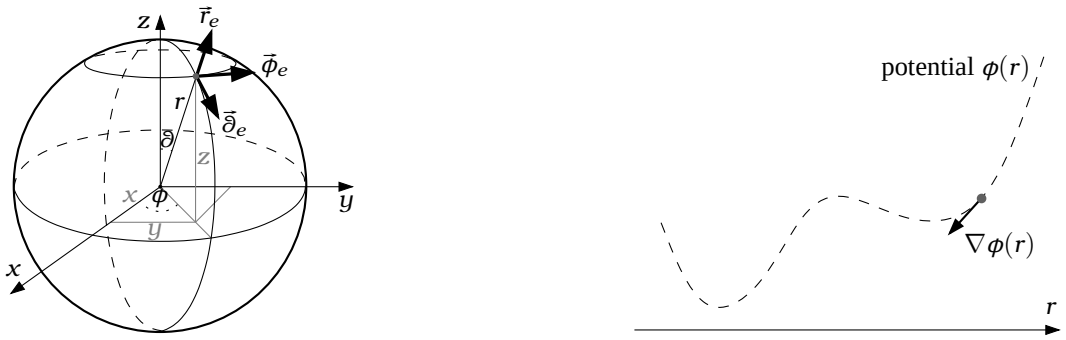



Figure 3.1: Calculation of the gradient for near field and far field parts.




(a) (b)

Figure 3.2: Spherical polar coordinates introduced on a sphere.

As seen in Chapter 2, computation is split up in two parts – the near field part, where all next neighbor interactions are computed, and the far field part for all remaining interactions. This structure is retained for gradient calculations and implies an implementation of the gradient computation on the same basis. Firstly, a derivative of the near field part, secondly the derivative of the far field part (See Figure 3.1).

## 3.1 Gradient Fundamentals

As there is a lot of discordance defining spherical coordinates especially for the symbols $\vartheta$ and $\varphi$, thus the conventions used in this thesis are explained briefly.

We define spherical coordinates as follows (See Figure 3.2). A point $p_k$ with spherical coordinates $(r_k, \vartheta_k, \varphi_k)$ has the following Cartesian coordinates:

$$x_k = r_k \sin\vartheta_k \cos\varphi_k, \quad y_k = r_k \sin\vartheta_k \sin\varphi_k, \quad z_k = r_k \cos\vartheta_k, \tag{3.1}$$

where $r_k \in [0,\infty)$, $\vartheta_k \in [0,\pi]$ and $\varphi_k \in [0,2\pi)$. The potential at point $\mathbf{r}_k$ due to all charges $q_l$ at position $\mathbf{r}_l$ is defined as:

$$\varphi(\mathbf{r}_k) = \sum_{l=1}^{N} \frac{q_l}{r_{kl}} \quad (k \neq l). \tag{3.2}$$

The respective Coulomb force acting on a particle with charge $q_k$ is then given by the gradient of $\varphi$:

$$\mathbf{F}(\mathbf{r}_k) = q_k \sum_{l=1}^{N} \frac{q_l}{r_{kl}^3} \mathbf{r}_{kl}. \tag{3.3}$$

## 3.2 Standard FMM Gradient

In section 2.1.1 it was shown that the FMM takes advantage of spherical harmonics to factorize the inverse distance. Since the input particle system is passed in Cartesian coordinates, the calculation of the Coulomb forces should be Cartesian likewise. However, the FMM is based on spherical coordinates. Thus, a transformation of the derivatives is necessary:

$$\begin{pmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{pmatrix} = \begin{pmatrix} \partial r/\partial x & \partial \vartheta/\partial x & \partial \varphi/\partial x \\ \partial r/\partial y & \partial \vartheta/\partial y & \partial \varphi/\partial y \\ \partial r/\partial z & \partial \vartheta/\partial z & \partial \varphi/\partial z \end{pmatrix} \begin{pmatrix} \partial/\partial r \\ \partial/\partial \vartheta \\ \partial/\partial \varphi \end{pmatrix}. \tag{3.4}$$

The Cartesian partial derivatives in spherical coordinates therefore are:

$$\begin{pmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{pmatrix} = \begin{pmatrix} \sin\vartheta\cos\varphi & \cos\vartheta\cos\varphi & -\sin\varphi \\ \sin\vartheta\sin\varphi & \cos\vartheta\sin\varphi & -\cos\varphi \\ \cos\vartheta & -\sin\vartheta & 0 \end{pmatrix} \begin{pmatrix} \partial/\partial r \\ \frac{1}{r}\partial/\partial\vartheta \\ \frac{1}{r\sin\vartheta}\partial/\partial\varphi \end{pmatrix}. \tag{3.5}$$

Numerical problems can occur for $r = 0$ and angles $\vartheta = 0$ as these values induce undefined expressions. Since it is possible that such a situation arises during the computation, special handling must be implemented. It is also possible to transform the spherical derivatives such that indefinite expressions vanish. In this case, not only $\partial/\partial\vartheta$ has to be derived, but both terms $\frac{1}{r}\partial/\partial\vartheta$ and $\frac{1}{r\sin\vartheta}\partial/\partial\varphi$ have to be transformed.

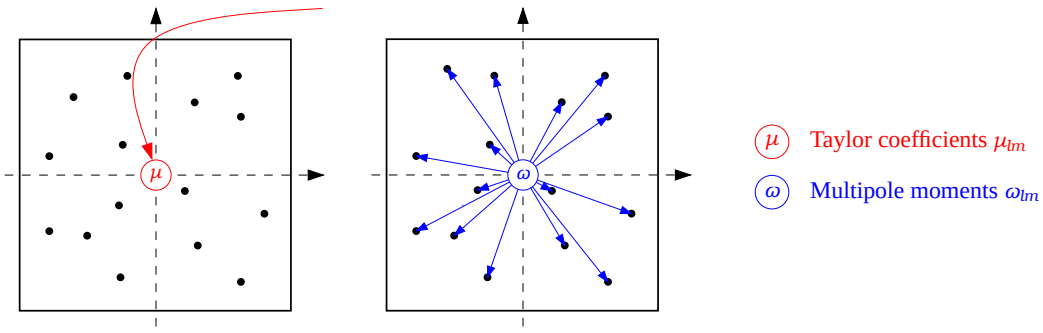
### 3.2.1 Standard Far Field Gradient

The Coulomb forces are defined as the gradient of the potential. As the potential is factorized in a multipole expansion and a local Taylor-like expansion, only the multipole part needs to be derived to obtain the Coulomb forces:

$$\frac{\partial E}{\partial(x_k, y_k, z_k)} = \sum_{l=0}^{L} \sum_{m=-l}^{l} \mu_{lm} \frac{\partial \omega_{lm}^k}{\partial(x_k, y_k, z_k)}. \tag{3.6}$$

A given multipole expansion, with:

$$\omega_{lm}^k = q r^l \frac{1}{(l+m)!} P_{lm} e^{-im\varphi}, \tag{3.7}$$




(a) Taylor-like expansion (b) Multipole expansion

Figure 3.3: Standard far field gradient: Both, the multipole gradient (b) of all local charges and the Taylor-like expansion (a) of all distant charges reside at the box center. Thus, the two contributions can be multiplied, yielding the far field gradient for each particle within the box.

must be derived for $\partial/\partial r$, $\frac{1}{r}\,\partial/\partial\vartheta$ and $\frac{1}{r\sin\vartheta}\,\partial/\partial\varphi$. Especially the latter is complicated, since derivatives $\frac{\partial\omega_{lm}}{\partial\varphi}$ do not contain parts of $\vartheta$, namely $1/\sin\vartheta$. By help of recurrence relations, the results are:

$$\frac{\partial\omega_{lm}}{\partial r} = q\,l\frac{r^{l-1}}{(l+m)!}P_{lm}e^{-im\varphi},$$

$$\frac{1}{r}\frac{\partial\omega_{lm}}{\partial\vartheta} = \frac{1}{2}q\frac{r^{l-1}}{(l+m)!}\left[(l-m+1)(l+m)P_{lm-1}-P_{lm+1}\right]e^{-im\varphi},$$

$$\frac{1}{r\sin\vartheta}\frac{\partial\omega_{lm}}{\partial\varphi} = -\frac{1}{2}q\frac{r^{l-1}}{(l+m)!}\left[(l-m+1)(l-m+2)P_{l+1m-1}+P_{l+1m+1}\right]i\,e^{-im\varphi}.$$

Hence, all derivatives are given in terms of the associated Legendre polynomials $P_{lm}$ only. Figure 3.3 illustrates this scheme.

### 3.2.2 Standard Near Field Gradient

For the near field part derivatives are less complicated since the inverse distance is not expanded and can be rewritten in Cartesian coordinates easily. Writing the total Coulomb

energy in Cartesian coordinates yields:

$$E = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{q_i q_j}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}}. \tag{3.8}$$

Re-writing as a double sum over all particles and demanding $i \neq j$ gives:

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{q_i q_j}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}}. \tag{3.9}$$

For $i \neq j$ the derivatives can be transformed to:

$$\frac{\partial E}{\partial x_k} = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} q_i q_j \left[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2\right]^{-3/2} (x_i - x_j)(\delta_{ik} - \delta_{jk}). \tag{3.10}$$

To separate the difference of the two delta functions $(\delta_{ik} - \delta_{jk})$, Equation (3.10) is transformed to:

$$\begin{aligned} \frac{\partial E}{\partial x_k} = &-\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} q_i q_j \left[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2\right]^{-3/2} (x_i - x_j)\delta_{ik} \\ &+\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} q_i q_j \left[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2\right]^{-3/2} (x_i - x_j)\delta_{jk}. \end{aligned} \tag{3.11}$$

Evaluating the delta functions leads to:

$$\begin{aligned} \frac{\partial E}{\partial x_k} = &\frac{1}{2} \sum_{j=1}^{N} q_k q_j \left[(x_k - x_j)^2 + (y_k - y_j)^2 + (z_k - z_j)^2\right]^{-3/2} (x_j - x_k) \\ &\frac{1}{2} \sum_{i=1}^{N} q_k q_i \left[(x_i - x_k)^2 + (y_i - y_k)^2 + (z_i - z_k)^2\right]^{-3/2} (x_i - x_k), \end{aligned} \tag{3.12}$$

with $j \neq k$ and $i \neq k$. Both sums can be combined by changing the index $j$ to $i$. Finally the derivative in direction $x_k$ is described by:

$$\frac{\partial E}{\partial x_k} = q_k \sum_{i=1}^{N} q_i \frac{x_i - x_k}{\left[(x_i - x_k)^2 + (y_i - y_k)^2 + (z_i - z_k)^2\right]^{(3/2)}}. \tag{3.13}$$

## 3.3 Alternative Far Field Gradient

Similarly to the standard implementation of the FMM gradient, the Taylor-like expansion resides at the center of each lowest level box. According to Equation (2.11) the far field part of the total Coulomb energy is given by:

$$E = \sum_{ibox} \sum_{l=0}^{L} \sum_{m=-l}^{l} \omega_{lm} \mu_{lm}.$$

In contrast to the former implementation, the Taylor-like expansion is not multiplied with the derivative of the multipole expansion at the box center to obtain the FMM gradient. Instead, the Taylor-like expansion is shifted to the particle position. Therefore, the operator **O** is applied (See section 2.1.7):

$$\mu_{lm}^{shift} = \sum_{j=l}^{L} \sum_{k=-j}^{j} O_{j-l\,k-m}\, \mu_{jk}.$$

A Taylor-like expansion $\mu_{jk}$ at box center shifted to particle position yields $\mu_{lm}^{shift}$. Correspondingly, all multipole derivatives are formed at particle position. This scheme is illustrated in Figure 3.4. Examining the derivatives, we find that all derivatives provide non-zero values for $l = 1$ only. The expression $r^{l-1}$ becomes zero for all $l \neq 1$, since this corresponds to evaluating the multipole expansion at particle position with $r = 0$. The complicated derivatives in Equation (3.8) take a simpler form:

$$\begin{aligned}
\frac{\partial \omega_{lm}}{\partial r} &= q \frac{1}{(1+m)!} P_{1m} e^{-im\varphi} \\
\frac{1}{r} \frac{\partial \omega_{lm}}{\partial \vartheta} &= \frac{1}{2} q \frac{1}{(1+m)!} \left[ (2-m)(1+m) P_{1,m-1} - P_{1,m+1} \right] e^{-im\varphi} \\
\frac{1}{r \sin\vartheta} \frac{\partial \omega_{lm}}{\partial \varphi} &= -\frac{1}{2} q \frac{1}{(1+m)!} \left[ (2-m)(3-m) P_{2,m-1} + P_{2,m+1} \right] \, i \, e^{-im\varphi}
\end{aligned}$$

Hence, we only need multipole coefficients with $l = 1$ and $m \in \{0, 1\}$. The derivatives can

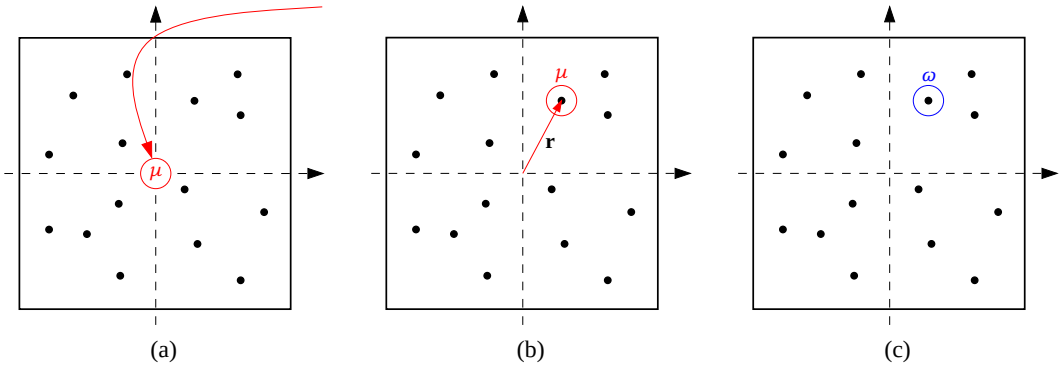



Figure 3.4: Alternative far field gradient – (a) In pass 2 remote multipole expansions are transformed to a local Taylor-like expansion at the center of each lowest level box. (b) The Taylor-like expansion is shifted to every particle position. (c) The multipole expansion is evaluated at particle position.

be simplified:

$$\begin{pmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \\ \frac{\partial E}{\partial z} \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{2}q\mathsf{Re}(\mu_{11}) & 0 \\ 0 & 0 & -\frac{1}{2}q\mathsf{Im}(\mu_{11}) \\ q\mathsf{Re}(\mu_{10}) & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathsf{Re}(\omega_{10}) \\ \mathsf{Re}(\omega_{11}) \\ \mathsf{Im}(\omega_{11}) \end{pmatrix}. \tag{3.14}$$

The computational complexity of operator $O$ decreases from $O(L^4)$ to $O(L^2)$ because only two multipoles are needed, namely $\omega_{10}$ and $\omega_{11}$[1].

## 3.4 Implementation Details

Our FMM implementation does not store multipole moments $\omega_{lm}$ for $m<0$, since this would waste memory with redundant information (See Figure 3.5). To obtain these moments, the following equation is used:

$$\omega_{l,-m} = (-1)^m \omega_{lm}^*, \tag{3.15}$$

[1] As the standard near field gradient does not contain a multipole approach, and thus can be seen as the classical gradient on a subset of particles, there will be no changes in the computation scheme. The alternative gradient algorithm only handles the far field part.

with $\omega^*_{lm}$ denotes the complex conjugate determined through the relation:

$$\omega^*_{lm} = [\mathsf{Re}\,(\omega)_{lm}) + \mathsf{Im}\,(\omega_{lm})]^* = [\mathsf{Re}\,(\omega_{lm}) - \mathsf{Im}\,(\omega_{lm})]\,. \tag{3.16}$$

The same holds for $\mu_{l,-m}$. The total Coulomb energy $E$ now can be written as:

$$E = \sum_{ibox} \sum_{l=0}^{L} \left[ \mathsf{Re}\,(\omega_{l0})\,\mathsf{Re}\,(\mu_{l0}) + 2 \sum_{m=1}^{L} \left(\mathsf{Re}\,(\omega_{lm})\,\mathsf{Re}\,(\mu_{lm}) - \mathsf{Im}\,(\omega_{lm})\,\mathsf{Im}\,(\mu_{lm})\right) \right]. \tag{3.17}$$

Taking the derivative of the last equation yields:

$$\begin{aligned} \frac{\partial E}{\partial (x,y,z)} \quad = \quad & \sum_{\text{ibox}} \sum_{l=0}^{L} \left[ \frac{\partial \mathsf{Re}\,(\omega_{l0})}{\partial (x,y,z)} \mathsf{Re}\,(\mu_{l0}) + \right. \\ & \left. 2 \sum_{m=1}^{L} \left( \frac{\partial \mathsf{Re}\,(\omega_{lm})}{\partial (x,y,z)} \mathsf{Re}\,(\mu_{lm}) - \frac{\partial \mathsf{Im}\,(\omega_{lm})}{\partial (x,y,z)} \mathsf{Im}\,(\mu_{lm}) \right) \right]. \end{aligned} \tag{3.18}$$

The first step in calculating the alternative gradient is to shift the Taylor-like expansion from the center of the box to each of the charges in any lowest box. Therefore, the Taylor-like expansion has to be multiplied with operator $O$. Equation (3.3) has to be transformed to be applicable, since we do not store coefficients with $k < 0$. All expressions with $k < 0$ must be transformed according to Equation (3.15). As only $\mu_{10}$ and $\mu_{11}$ are needed for calculating the alternative gradient, the shifted Taylor-like expansion for $\mu_{10}$ is represented by:

$$\mu_{10}^{shift} = \sum_{j=1}^{n} \left[ O_{j-1,0}\mu_{j0} + \sum_{k=1}^{j} \left( O_{j-1,k}\mu_{jk} + (-1)^{2k} O^*_{j-1,k}\mu^*_{jk} \right). \right] \tag{3.19}$$

However, $O_{j-1,j}$ is always zero, hence the inner sum only has to be taken for indices until $j-1$:

$$\mu_{10} = \sum_{1}^{n} \left[ O_{j-1,0}\mu_{j0} + \sum_{k=1}^{j-1} \left( O_{j-1,k}\mu_{jk} + (-1)^{2k} O^*_{j-1,k}\mu^*_{jk} \right) \right]. \tag{3.20}$$

For $\mu_{11}$ we continue likewise:

$$\mu_{11} = \sum_{j=1}^{n} \left[ O_{j-1,-1}\mu_{j0} + \sum_{k=1}^{j} \left( O_{j-1,k-1}\mu_{jk} + (-1)^{2k+1} O^*_{j-1,k+1}\mu^*_{jk} \right) \right]. \tag{3.21}$$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $j$ | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| $k$ | 0 | 0 | 1 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 | 5 |

Figure 3.5: The multipole moments $w_{jk}$, and respectively the coefficients of the Taylor-like expansion $\mu_{jk}$, are stored in memory with regard to their index.

By splitting the inner sum into two parts we get:

$$\mu_{11} = \sum_{j=1}^{n} \left[ -O^*_{j-1,-1}\mu_{j0} + \sum_{k=1}^{j} \left(O_{j-1,k-1}\mu_{jk}\right) + \sum_{k=1}^{j-2} \left(O^*_{j-1,k+1}\mu^*_{jk}\right) \right]. \tag{3.22}$$

To combine the shifts for $\mu_{10}$ and $\mu_{11}$, further transformations take place:

$$\begin{aligned} \mu_{10} \;=\; & O_{00}\mu_{10} + O_{10}\mu_{20} + O_{11}\mu_{21} + O^*_{11}\mu^*_{21} + \\ & \sum_{j=3}^{n} \Bigg[ O_{j-1,0}\mu_{j0} + \\ & \sum_{k=1}^{j-2} \left(O_{j-1,k}\mu_{jk} + O^*_{j-1,k}\mu^*_{jk}\right) + O_{j-1,j-1}\mu_{j,j-1} + O^*_{j-1,j-1}\mu^*_{j,j-1} \Bigg], \end{aligned} \tag{3.23}$$

$$\begin{aligned} \mu_{11} \;=\; & O_{00}\mu_{11} - O^*_{11}\mu_{20} + O_{10}\mu_{21} + O_{11}\mu_{22} + \\ & \sum_{j=3}^{n} \Bigg[ - O^*_{j-1,1}\mu_{j0} + \\ & \sum_{k=1}^{j-1} \left(O_{j-1,k-1}\mu_{jk} - O^*_{j-1,k+1}\mu^*_{jk}\right) + O_{j-1,j-2}\mu_{j,j-1} + O_{j-1,j-1}\mu_{jj} \Bigg]. \end{aligned} \tag{3.24}$$

Now, both $\mu_{10}$ and $\mu_{11}$ can be used to calculate the alternative gradient in one loop over $j$. Thereby, the number of load and store operations is reduced, hence the calculation can be performed faster.

## 3.5 Comparing Both Algorithms

The alternative algorithm takes a simpler form and avoids complicated derivatives. The alternative implementation leads to 20% faster results for the gradient subroutine. This improve-

| Method | Gradient I | Gradient II | Gradient III* |
|---|---|---|---|
| total time (seconds) | 284.89 | 283.20 | 277.62 |
| instructions | $7.480 \cdot 10^9$ | $7.499 \cdot 10^9$ | $5.960 \cdot 10^9$ |
| divisions | $86.247 \cdot 10^6$ | $73.955 \cdot 10^6$ | $73.937 \cdot 10^6$ |
| load/store | $2.593 \cdot 10^9$ | $2.558 \cdot 10^9$ | $1.662 \cdot 10^9$ |
| TLB[1] misses | 46470 | 16998 | 26903 |
| L1 cache[2] hit rate | 81.421% | 94.359% | 90.124% |
| FMA[3] percentage | 37.133% | 67.193% | 61.983% |
| instructions (rel.) | 100.00% | 100.25% | 79.68% |

* Improved alternative gradient II algorithm with reduced number of instructions.

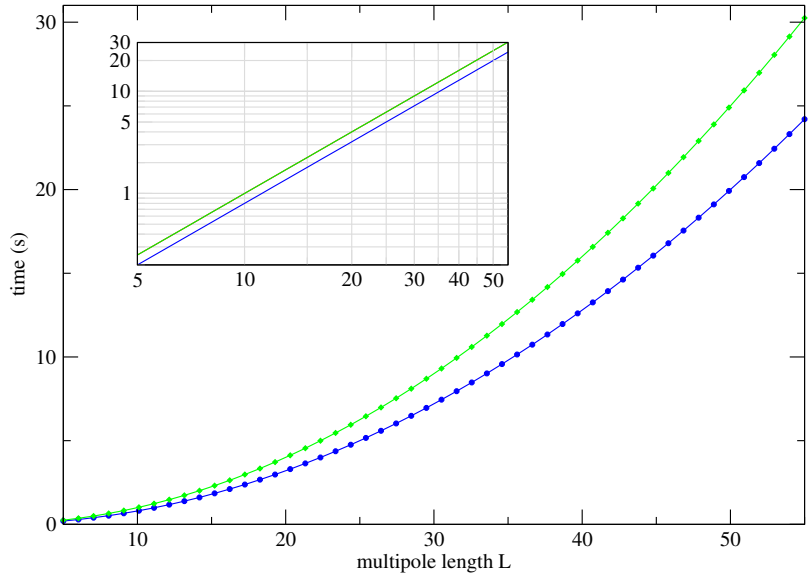
[1] TLB: Translation Lookaside Buffer

[2] L1 cache: Level 1 cache

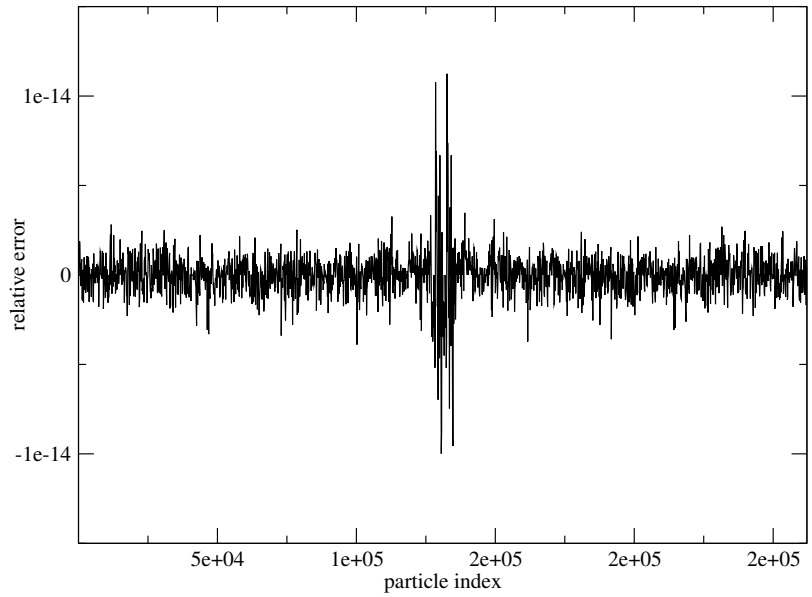
[3] FMA: Fused Multiply and Add

Table 3.1: Hardware performance monitoring for both gradient algorithms.

ment is independent from the number of multipoles used and does not depend on the given accuracy. Even very high accuracy calculations with 20 and more multipoles benefit from this implementation. Table 3.1 compares different gradient algorithms. The standard FMM gradient, the alternative FMM gradient and the improved alternative FMM gradient have been instrumented using *libhpm* [35] hardware counter to measure several runtime properties. The improved alternative gradient differs in the subroutine's loop structure. Both $\mu_{10}$ and $\mu_{11}$ are calculated using one loop only. The number of load and store operations is reduced and the code performes 20% faster. The number of instructions is reduced by 20% and load and store operations reduce to 64%. Without this improvement the alternative algoritm performs as fast as the standard algorithm. The results are shown in Figure 3.6a and 3.6b.




(a) Gradient computation shows the expected non-linear $L$ scaling. The subplot represents a log-log plot of the data.




(b) Both gradient implementations yield the same results with relative error in the magnitude of the machine precision.

Figure 3.6: (a) The figure shows the scaling of the gradient algorithms with regard to multipole length $L$. However, the $L$-scaling does not show the theoretical cubic behavior. Instead, the plot shows a quadratical dependency of $L$. The non-linear fit yields 1.98 as exponent. This dependency can also be seen in Table 2.3 on page 22. (b) The figure shows the relative error of the alternative gradient algorithm in relation to the standard implementation.

# 4 Parallelization of the FMM

The solution of the $N$-body problem is considered to be a "Grand Challenge Problem". There is a broad field of scientific applications ranging from astronomy over molecular dynamics, plasma physics and fluid dynamics. Grand Challenge Applications (GCA's) usually require state-of-the-art massively parallel computers. One could e.g. think of protein folding, with about 1 million particles and more than 1000 time steps. Without using parallel computers and algorithms, solving this problem would be out of reach. A parallel version of the FMM offers a viable approach to further expand the scale of these scientific applications. Using such an implementation in molecular dynamics codes allows to increase the number of particles to be simulated and to perform more time steps. This chapter illustrates a first approach towards a parallel version of the FMM.

## 4.1 Programming Models

Parallel software development introduces several new problems not encountered during sequential programming, namely: data partitioning and distribution, communication, synchronization and load balancing. Only two models will be introduced briefly to establish the fundamentals of this area.

### 4.1.1 Message Passing

Message passing is a method of communication between processes. One process sends data and another process receives the data. When writing parallel applications using message passing, the programmer still has to develop a significant amount of program code to handle many tasks of the parallelization, such as:

- data partitioning and distribution,
- the communication and synchronization between processes, and
- mapping of processes onto processors.

Common message passing libraries for scientific computation are e.g. MPI [20] or PVM [16].

### 4.1.2 Distributed Shared Memory (DSM)

In contrast to message passing, in a DSM environment a process fetching data does not need to know its location in the remote process' memory; the library finds and fetches it automatically based on a global index. While scalable parallel machines are mostly based on distributed memory, one may find it easier to use contiguous memory indexing for parallel programs, which is provided by a shared memory programming model. A special shared memory style environment named *Global Arrays Toolkit* is presented in section 4.4. Others are e.g. Posix Threads [8] or System V Shmem [26].

### 4.1.3 Design

There is no simple recipe for designing parallel algorithms. We use the approach of data parallelism. Other approaches can be found in [41, 33], but will not be considered here. However, it is possible to split up the design process into four distinct stages :

**Partitioning** Starting from the sequential code the data it operates on can be decomposed into several smaller chunks. This method, the decomposition of the data associated with the problem, is called domain/data parallelism.

**Communication** Communication is necessary to transfer data between cooperating processes. Usually the design of parallel algorithms aims to reduce the communication volume, as transferring data between different processes is time consuming.

**Agglomeration** The communication structure defined by the previous step is not optimal most likely. For performance reasons, some individual communications may be bundled together into a single communication. This helps to reduce the overall communication overhead. However, combining messages does not come for free, computational costs will increase.

**Load Balancing** The goal of a parallel problem is a maximal utilization of the system resources, while keeping communication costs minimal. When it is not possible to perform

static load balancing at compile time, dynamic load balancing at runtime can improve performance.

## 4.2 Parallel Programming Paradigms

The type of parallelism inherent in the problem and available computing resources determine the choice of the paradigm. The structure of the program or the data binds to a special kind of parallelism. The most popular approaches are described below.

1. Task/Farming (Master/Slave)
2. Single program multiple data (SPMD)
3. Divide and Conquer

### 4.2.1 Task-Farming: Master/Slave

The task-farming approach subdivides the computational resources into two entities: master process(es) and multiple slave processes (See Figure 4.1a). The master decomposes the main problem into smaller tasks and distributes these among the slave processes. The final result of the computation is assembled by the master after collecting the partial results from the slave processes. The slave process only has to receive a message containing a task, process the task and send a message back to the master with the results. As the slaves are controlled by the master there usually is no communication between slaves. Only the master and slaves communicate. Load balancing may either be static or dynamic and is implemented in the master process.

For the static case, the distribution is forwarded to the slaves at the beginning of the computation. Thus, the master can participate in the computation. Dynamic load-balancing is more suitable,

- when prediction of the execution time is not possible, or
- when less processors than tasks are available, or
- when the number of tasks is not known before the communication.

The dynamic scheme has several advantages. The parallel application can adjust to changes in the load of the processors. Also, it is possible to compensate a total outage of some slave

processes. However, the centralized role of the master process puts some restraints on the scalability of the program. For a large number of processes, the communication between master and slaves will use a large amount of time. To overcome this restraint and improve the scalability, a single master process can be replaced by a set of masters, each of them controlling a subset of slaves.

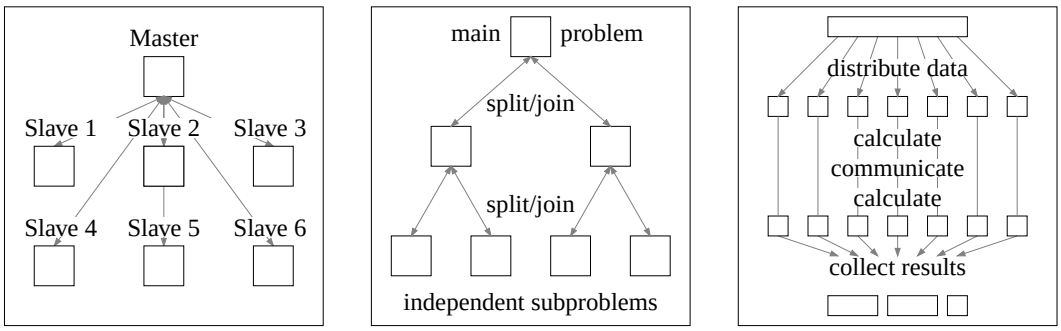
### 4.2.2 SPMD

In the SPMD paradigm each process executes the same code, but works with different parts of the data. This is also suitable for the FMM. The application splits the data among the processors that are available (See Figure 4.1c). Due to the splitting it is also called geometric parallelism, domain decomposition, or data parallelism. The underlying regular geometric structure and the spatially limited interactions in the FMM and many physical problems make them suitable for the SPMD paradigm. Communication is generally performed with neighbor processes, and the associated communication load is proportional to the size of the boundary elements, while the computation load will be proportional to the volume of the elements. Thus, the communication pattern is highly structured. The data may initially be loaded by each process. For homogeneous systems and properly distributed data the parallel SPMD application may show up very efficient. An inhomogeneous distribution can be handled via a load balancing scheme. Thus, the program can adapt the data distribution layout during run-time.

### 4.2.3 Divide and Conquer

This approach partitions the problem into several independent subproblems. These subproblems can be solved by the involved processes without communication. The results of the subproblem can be merged and thus form the final result. This scheme is already known from sequential divide and conquer algorithms. In a parallel implementation, the subproblems can be solved independently and simultaneous. Thus, the scheme can be described in three steps (See Figure 4.1b):

- split,
- compute,
- merge.




(a) Master/Slave (b) Divide and Conquer (c) SPMD

Figure 4.1: Different parallel programming paradigms are available for parallelization.

## 4.3 Implementation

The parallelization schemes presented, offer a broad range of possibilities, but not all of them are suitable for the FMM. A main issue in parallelizing the FMM is the lack of memory. As shown in Table 4.2 on page 47, a billion particles would require about 100 GB of memory. Thus, it is impossible to store redundant data on every process. Data must be distributed and can not be replicated. Each process is allowed to store a fraction of all particles only. Therefore, the task-farming approach is not suitable. The master – a single process – would need to store the entire simulation data. All other processes – the slaves – would receive their fraction.

A more suitable approach with less data redundancy is the divide and conquer paradigm. By employing, the parallelization takes advantage of the tree-like structure of the FMM. The tree is split up among the processes. Interacting boxes at the edge are the only redundant data stored. For the lowest level no communication between the processes takes place. However, for higher levels of the tree, communication is necessary.

To completely avoid redundant data storage the SPMD scheme is preferable. In this scheme, one process stores only a fraction of the full data. All interacting boxes not contained in the local memory have to be fetched from remote processors. This scheme allows a minimal memory consumption, but comes at the costs of increased communication. As it is our goal to handle very large particle systems, the parallelization of the FMM was to limit the system size by the computing power, but not the available memory. Thus, the SPMD paradigm was chosen for parallelization.

Furthermore, the structure of the sequential FMM had to be preserved in the parallel version. Therefore, data structures of the sequential implementations are represented in the same manner via distributed shared memory for the parallel version. All distributed arrays are generated as global arrays, setup and data management are performed by the GA library. All communication can be implemented one-sided without explicitly specifying send and receive operations on the remote process. Also load balancing schemes and asynchronous communication can be implemented more efficiently.

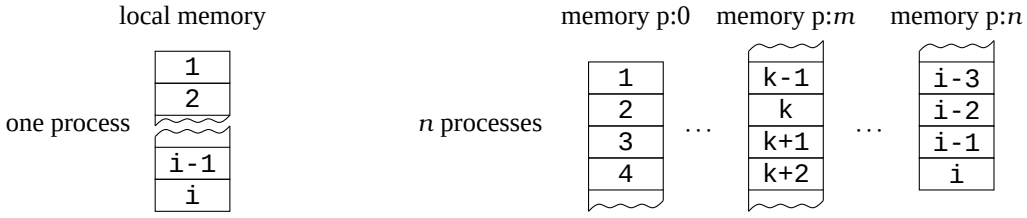
## 4.4 The Global Arrays (GA) Toolkit

### 4.4.1 Description

The *Global Arrays Toolkit* [28] provides a shared-memory programming interface for distributed-memory computers. Each process can asynchronously access logical blocks of physically distributed data, without explicit cooperation with the host process. Global Arrays have been designed to offer a shared-memory environment on top of a message-passing programming model. It provides the programmer both with shared-memory and message-passing functionality. Global Arrays are compatible with the Message Passing Interface (MPI).

Data is stored in global arrays and can be used as if it were stored in shared memory. All details of the data distribution, addressing, and data access are encapsulated in a global array object. GA logically divides shared data structures into local and remote portions. Any process can access a local portion of the shared data directly like any other data in the local memory. Access to other portions of the shared data must be done through GA library calls.

The shared memory operations `get`, `put`, `scatter` and `gather` are used to access the remote data. The communication is implemented using one-sided communication. It is possible to generate both regular or irregular data distribution patterns.
It is not necessary to deal with addresses of the distributed data, as the GA toolkit offers an index-based interface for access. Additionally, it is not necessary to specify the target process id when executing a remote memory operation. GA will find out by itself, where the referenced data resides. However, it is possible to request information of the data distribution.




(a) Sequential data distribution

(b) Parallel data distribution

Figure 4.2: (a) Continuous memory can be accessed directly on one process. (b) Global data is distributed among $n$ processes. Data has to be accessed by GA routines.

### 4.4.2 Relevant GA Operations

To parallelize the FMM, only a few functions from the GA library are necessary. This section outlines all functions [29] used to initialize the GA environment and perform remote operations on the global data.

#### GA Operations - Properties

**collective** operations require all processes to make the call.

**local** operations are local to each process and do not require communication.

**atomic** operations have mutual exclusion built in. Concurrent read/write operations are prevented by the GA library.

#### GA Process Information

```
integer function ga_nodeid()
integer function ga_nnodes()
```

The functions `ga_nodeid` returns the process ID and `ga_nnodes` returns the total number of processes.

#### One-sided Communication, Blocking

```
subroutine nga_acc(hdl,loidx,hiidx,buf,ldim,scale)
subroutine nga_put(hdl,loidx,hiidx,buf,ldim)
subroutine nga_get(hdl,loidx,hiidx,buf,ldim)
```

The *Global Arrays Toolkit* provides one-sided operations that allow access to global arrays without implicit interaction with the process holding the requested data. To perform remote

read and write operations, the procedures `nga_get` and `nga_put` are required. With help of `nga_acc`, a remote update can be applied to specific parts. This subroutine adds the data moved to the target process, rather than replacing the data. The operation is *atomic*. GA assures that multiple processes accessing the same patch will be handled correctly and in a consistent way.

### One-sided Communication, Non-blocking

```
subroutine nga_nbacc(hdl,loidx,hiidx,buf,ldim,scale,waithdl)
subroutine nga_nbput(hdl,loidx,hiidx,buf,ldim,waithdl)
subroutine nga_nbget(hdl,loidx,hiidx,buf,ldim,waithdl)
subroutine nga_nbwait(waithdl)
```

The non-blocking implementation provides the same features as the blocking one, however non-blocking operations return before data transfer is completed. To finalize the operation and reuse the variables involved, `nga_nbwait` has to be called.

### Local Access to GA

```
subroutine nga_distribution(hdl,myid,loidx,hiidx)
subroutine nga_access(hdl,loidx,hiidx,ptr,ldim)
subroutine nga_release(hdl,loidx,hiidx)
```

To access the local part of the global array directly without `nga_get` or `nga_put` calls, the toolkit allows direct access via an integer handle that acts like a pointer to the local address. Therefore, `nga_distribution` finds the local part of the global array. Access is established via a `nga_access` call. To release access one has to call `nga_release`.

### Initialization, Termination

```
subroutine ga_initialize()
subroutine nga_create(atype,ndims,dims,aname,chunk,hdl)
subroutine ga_fill(hdl,value)
subroutine nga_destroy(hdl)
subroutine ga_terminate()
```

After setting up the message-passing library (e.g. MPI), the GA library is initialized by `ga_initialize`. Creating new arrays is done via the `nga_create` subroutine. This creates a global array, which is regularly distributed over all processes. It can be up to seven dimensional. Setting initial values is done via the `ga_fill` subroutine. Global arrays can be destroyed by calling `ga_destroy`. To terminate the GA program, the subroutine `ga_terminate` is called.

### Synchronization

```
subroutine ga_sync()
```

A global barrier can be established via the `ga_sync` command. This collective operation synchronizes all processes and ensures that all global operations started before the `ga_sync` are completed after the call.

### Global Operations

```
subroutine nga_read_inc(hdl,index,value)
subroutine ga_igop(MT_INT,var,length,op)
subroutine ga_dgop(MT_INT,var,length,op)
```

The function `nga_read_inc` remotely updates a particular element in the global array. It can be seen as a global counter. This operation is atomic.

## 4.5 Sequential FMM Data Layout

The FMM consists of five passes. (See section 2.1) The most time-consuming parts are pass 2 (See section 2.1.3 on page 9) and pass 5 (See section 2.1.6 on page 12). Pass 2 calculates the far field interactions via multipoles. Pass 5 calculates the near field interactions via a direct summation. Both pass 2 and pass 5 consume more than 90% of the total computation time. (See Table 4.1 for details.) The data for a FMM run can be divided into

- input data,
- output data and
- auxiliary data.

The input data includes the charge and the position of the simulation particles in three dimensions and the designated error bound. Output data are the total Coulomb energy, the Coulomb forces for each particle and the Coulomb potential for each particle. During the calculation the FMM needs additional auxiliary data. Each particle is assigned a certain box. The box vector stores the box number for the particles. To accelerate access to neighbor boxes, this vector is stored twice with a different bit order. Thus, it is not necessary to calculate the index of neighboring boxes over and over again. Furthermore, a list is required to store the multipole and Taylor-like expansion for each box. Since it is essential to establish an order in the input data, an additional sort vector is stored to allow re-ordering after the FMM finishes.

| Pass | Status | Dependencies | Runtime |
|---|---|---|---|
| Setup | - | - | 2.20% |
| Pass 1 | sequential | local & tree | 1.47% |
| Pass 2 | parallel | neighbor & tree | 29.53% |
| Pass 3 | sequential | tree | 1.00% |
| Pass 4 | sequential | local | 2.20% |
| Pass 5 | parallel | neighbor | 63.80% |

Table 4.1: Approximate computation time of the single passes.

## 4.6 Parallel FMM Data Layout

Firstly, the passes 2 and 5 were considered for the parallelization. These passes are the most time-consuming parts and take more than 90% of the computation time. Therefore, it is appropriate to begin parallelization only for these parts. For a proper scalability at high processor numbers, it is important to parallelize all parts of the FMM and reduce the number of sequential parts in the code to a minimum. However, this is beyond the scope of this thesis. As described in section 4.3, the FMM data has to be partitioned by a domain composition and distributed to the processes. The calculations on each process are performed without interleaved communication. Network operations are necessary for overlapping boxes or neighbor boxes only. An overlapping box is shared by two or more processes. Thus, one of the processes involved has to fetch data from remote process(es). The same holds for neighbor boxes that might reside on another process.

To take full advantage of the memory that is available all data structures with a size depending on the particle or box number are distributed among the processes. Thus, all the data shown in Table 4.2 has to be distributed. The distribution is carried out by using space filling curves (SFC) [1]. Other FMM implementations used similar techniques [18, 30].

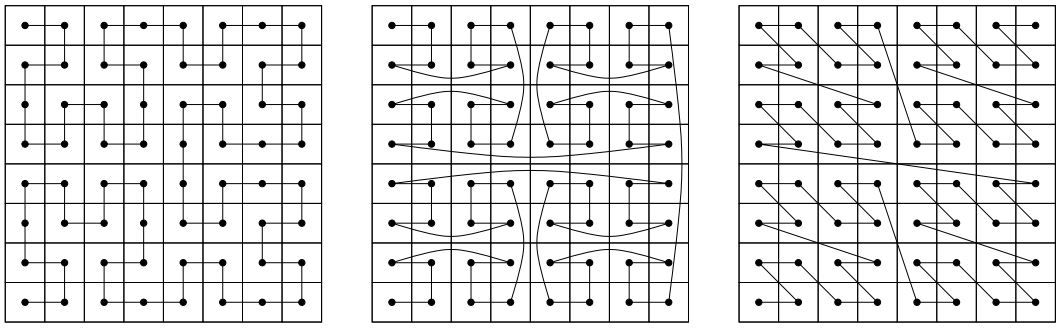
### 4.6.1 Space-filling Curves

The ordering of the data in memory is of little interest for the sequential implementation of the FMM. All the data can be accessed directly in memory. However, for the parallel version, every process only can access its local data directly. Thus, the workload of the problem has

| Type | Name | Data Type | Elements | Size per Entry |
|---|---|---|---|---|
| input | coordinates | `double` | $3n$ | 8 bytes |
| input | charges | `double` | $n$ | 8 bytes |
| input | error bound | `double` | 1 | 8 bytes |
| output | Coulomb energy | `double` | 1 | 8 bytes |
| output | Cartesian forces | `double` | $3n$ | 8 bytes |
| output | Coulomb potential | `double` | $n$ | 8 bytes |
| aux. | box vector | `integer` | $n$ | 8 bytes |
| aux. | box scratch vector | `integer` | $n$ | 8 bytes |
| aux. | box sort vector | `integer` | $n$ | 8 bytes |
| aux. | multipole expansion | `complex` | $\frac{1}{2}(L+1)(L+2)b$ | 16 bytes |
| aux. | Taylor expansion | `complex` | $\frac{1}{2}(L+1)(L+2)b$ | 16 bytes |

Table 4.2: Memory requirements for input, output and auxiliary data necessary for the FMM; 100 GB memory for one billion particles ($n = 8^{10}$) and five multipoles ($L = 5$).Variable $b$ represents the number of non-empty boxes and therefore depends on the particle distribution.

to be distributed evenly in order to use parallelism efficiently. Otherwise some processes will become idle and therefore do not participate in the computation any longer. For a large number of processes it is very important to implement a proper static load balancing. Once the data is distributed unevenly among the processes, a redistribution becomes very inefficient and expensive, because it makes another communication step necessary. Therefore, one has to prevent an uneven distribution in advance. An even distribution assures a consistent computation time for all processes. Figure 4.3 illustrates several ways to partition the simulation space. Three examples are shown; a distribution along a Hilbert curve, a distribution along a Gray curve and a distribution along a Morton curve. These curves differ from each other with regard to jumps and locality of the curve. A common property of all space-filling curves [27] is that every box is passed only once. The 2D examples can be extended to three or more dimensions as well. Thereby, the multidimensional structure is transformed to one dimension. Through this, it is easy to partition the 3D structure into $p$ equal parts for $p$ processes. Then, these sections can be distributed among the processes. The space-filling curve should not contain jumps to preserve data locality. The Hilbert curve is most suitable, since it does not



(a) Hilbert curve (b) Gray curve (c) Morton curve

Figure 4.3: Several space-filling curves. For simplicity in two dimensions.

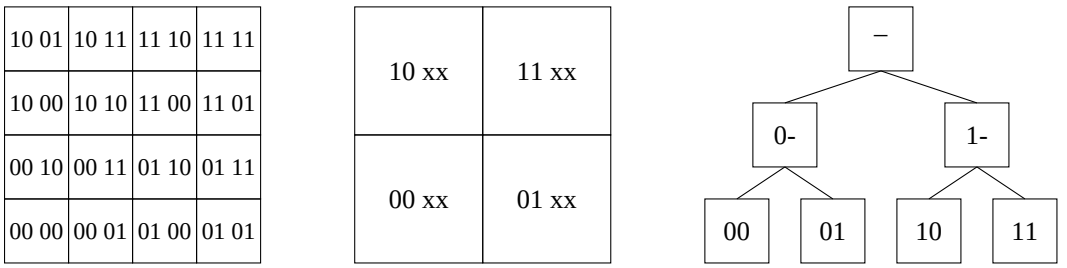
| (x,y,z) | Dimension | | | Mixing | SFC Index |
|---|---|---|---|---|---|
| | x | y | z | | |
| (0,2,1) | **000** | 010 | *001* | **0**0*0***0**1*0***0**0*1* | 17 |
| (3,0,7) | **011** | 000 | *111* | **0**0*1***1**0*1***1**0*1* | 111 |
| (7,2,0) | **111** | 010 | *000* | **1**0*0***1**1*0***1**0*0* | 308 |

Table 4.3: Morton space-filling curve and bit mixing in three dimensions. Each box is represented by its coordinates $(x, y, z)$. This three dimensional representation can be transformed to a Morton ordered one-dimensional box index (SFC index).

contain any jumps. The next box along this curve is a direct neighbor every time. However, a Morton ordering of the data was chosen for the parallelization of the FMM. This curve has advantages for the FMM structure itself. Table 4.3 shows how one can obtain corresponding neighbor boxes easily. A parent box of a certain box can be determined using a bit shift on the box index. Thus, the navigation in the FMM tree from top to bottom and the other way round is very efficient.

### 4.6.2 Data Distribution

The input data arrays have to be sorted according to a box vector. Thus, the charges of the particles and the coordinates are ordered using the same SFC used for the box vector. This configuration assures the same indexing and distribution for all input arrays. To obtain the




(a) 2D child boxes (b) 2D parent boxes (c) 1D FMM tree with child and parent boxes

Figure 4.4: Morton ordering the data has several advantages. The SFC can be used to statically distribute the array among the processes [1]. (a) and (b) A bit shift enables access to child or parent boxes. On the other hand, tree traversal is kept simple (c), since it can be performed by shifting bit positions.

charge or position of particle $k$, the box index $k$ can be used to look up the data. To obtain locality, the charge and position has been distributed like the box index.

### 4.6.3 Parallel Subroutines

The granularity of the parallel version of the FMM is at box level. Thus, all operations that require full boxes remain sequential. A deeper parallelism was not introduced, as calculation with only a few boxes would lack performance. On the other hand, even for small systems ($< 10^5$ particles) enough boxes for distribution exist. For operations on the particle level parallel subroutines are necessary. The setup of the box vector or sorting the data into the boxes are two examples, where the sequential subroutines have to be replaced by parallel versions.

#### Box Vector

The management of the particles in the FMM is performed via the box vector. The length of this vector corresponds to the number of particles. Each particle is assigned an unique box number on each level (See Figure 4.4). This number is represented as a binary key. The setup of the box vector is shown in Figure 4.5. Since the box structure is defined by a Morton ordered space-filling curve, it is possible to access neighbor data very efficiently. Thus, a bit shift to the right yields the parent box. A shift to the left admits access to the child boxes. For example, a box number of 42 would be 101010 in binary notation. A bit shift to the right

| adress | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 7 | 7 | 9 | 9 | 9 | 9 | 13 | 13 | 13 | 13 |

Figure 4.5: The distributed box vector has redundant information. The viewed process contains elements starting from address 'C' to address 'S'. The elements '... A,B' and 'T ...' reside on neighbor processes.

| adress | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | -3 | -2 | -1 | 4 | -3 | -2 | -1 | 6 | -3 | -2 | -1 | 7 | -1 | 9 | -3 | -2 | -1 | 13 | -4 | -3 | -2 |

Figure 4.6: Box vector in skip vector form. To establish fast access to the elements of this vector all redundant informations are replaced by jump positions to the next box index. Thus, the search for a certain box index becomes faster, since only a few search steps have to be performed.

would give 101 and hence a box number of 5.

### Sorting

The FMM needs sorted data in many places. For setting up the FMM tree, all particles have to be sorted into the associated boxes. This sorting takes place at every tree level. Starting at level two, the particle position and charge are already pre-sorted. Subsequent sorting steps can benefit from this pre-sorting. The sorting has to scale linearly with the number of particles, since we want an implementation that scales linearly. Radix sort offers this property. The parallel FMM uses parallel radix sorting. The sorting is done in-place to ensure efficient memory usage. Here, in-place means that no additional memory is required that depends on the size of the data to be sorted. The sorting algorithm is allowed to take up a constant amount of memory only. The sorting subroutines are encapsulated in a C library and therefore can be adapted to the FMM via preprocessor directives [23] in a flexible way. Sorting is done with respect to the box vector. Starting from box number 1, all particles are sorted into the boxes. Sorting stops at the lowest level. No further sorting is done, since all intra box interactions are computed together. Finishing the computation, all calculated data is sorted back to fit the input data.

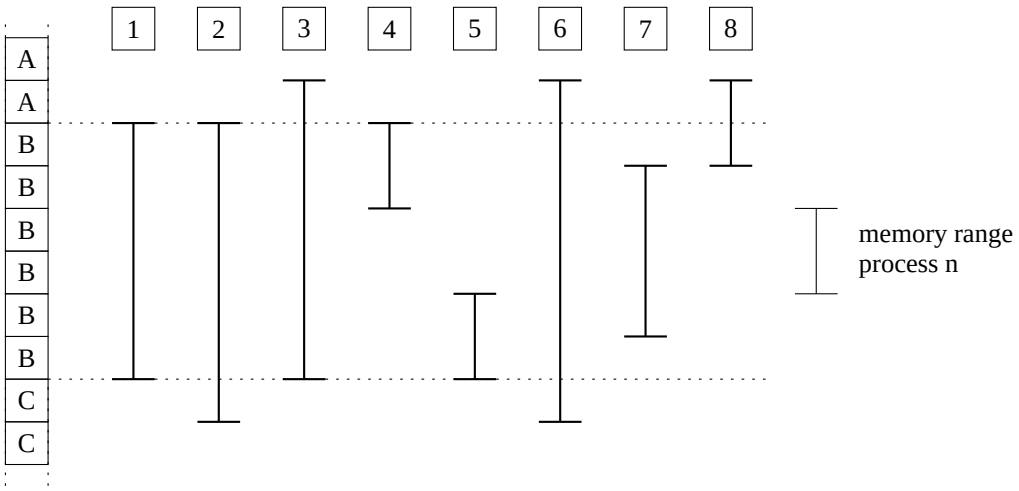



Figure 4.7: Parallel skip-box vector. Elements of one and the same box can overlap several processes. Therefore, communication is necessary to setup the box vector in parallel.

## 4.7 Locality & Global Operations

Introducing parallelism becomes easier when using a distributed shared memory approach. Every direct access to distributed data has to be replaced by a call to `get` for reading and a call to `put` for writing. Therefore an additional buffer has to be set up. This buffer holds the input and output data of the local calculation. After finishing the calculation, the buffer can be written to the global data and new data can be collected from the global data. However, this scheme is not optimal. Even local data would be buffered and therefore generate computational overhead in comparison to direct access. Performing several million read and write operations via put/get would slow down the algorithm (See Table 4.5). Therefore, most of the sequential functions have been parallelized in two versions. A complete local operation is done via direct variable access. An operation which is not completely local, will use the second function, which uses put/get calls to read or write the data. This scheme is illustrated in Listing 4.1.

### Pass 5

The parallel design of the FMM described in this section considers pass 5 as example. In the fifth pass, the near field interaction is calculated. The sequential version of pass 5 consists of two sub-functions. The first one computes the intra box interactions on the lowest level.

| Variable | Explanation | Value |
|---|---|---|
| `bbox(1,level,myid)` | global address of first local box | `C` |
| `bbox(2,level,myid)` | global address of last local box | `S` |
| `bbox(3,level,myid)` | first local box number | `3` |
| `bbox(4,level,myid)` | last local box number | `13` |
| `bbox(5,level,myid)` | global address of first local starting box | `D` |
| `bbox(6,level,myid)` | global address of last local ending box | `Q` |
| `bbox(7,level,myid)` | box number of first local starting box | `4` |
| `bbox(8,level,myid)` | box number of last local ending box | `9` |
| `bbox(9,level,myid)` | `yes`, if no local starting box exists | `no` |
| `bbox(10,level,myid)` | number of local charges | `17` |
| `bbox(11,level,myid)` | first local last box number | `3` |

Table 4.4: Box element information is shared among the processes prior to setup the skip-box vector to minimize communication.

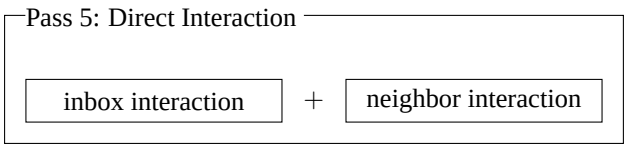



Figure 4.8: Sequential pass 5.

The second one computes the box to neighbor-box interactions on the lowest level. Figure 4.8 illustrates this scheme.

However, the parallel version has to handle distributed data. One can assume that the number of lowest level boxes is higher than the number of processes. Thus, a single process holds more than one lowest level box. An overlapping box can occur between two neighbor processes only. With $p$ processes, there are $p-1$ overlapping boxes. However, it is possible for a special configuration that one box extends more than two processes. This particular case is implemented as well, but will not be discussed here. Since parallelism ends at box level, divided boxes must be communicated and this will slow down the computation.

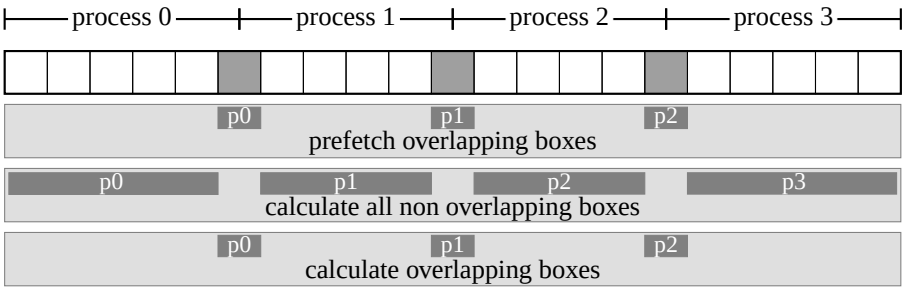



Figure 4.9: All intra-box interactions are calculated without communication. Only the computation of intra-box interaction for boxes shared by two or more processes need communication. These overlapping boxes are prefetched. Afterwards all non-overlapping intra-box interactions are calculated. Finally the prefetched boxes are used to calculate the remaining interactions. This scheme is shown for four processes exemplarily.

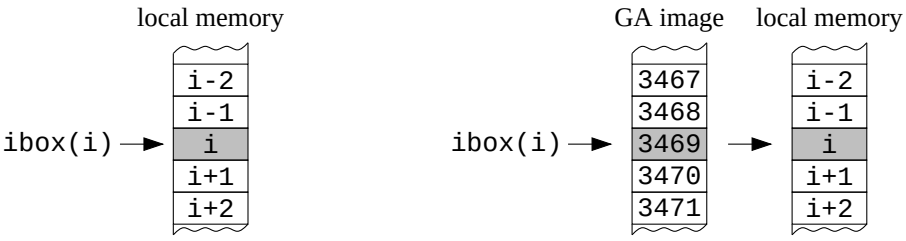



Figure 4.10: Local data can be accessed directly without using GA's get/put routines.

```
function ibox(a)
  if a == local then
    buf=ibox1loc(localaddress)
  else
    call get(a(globaladdress))
  endif
  ibox=buf
end function ibox
```

Listing 4.1: Wrapper function to access local and remote portions of the box vector.

Therewith, it is obvious that all intra box interactions except for overlapping boxes can be computed locally. Every process needs its local data only. Overlapping boxes can be distributed prior to computation, and thus reduce idle times (See Figure 4.9).

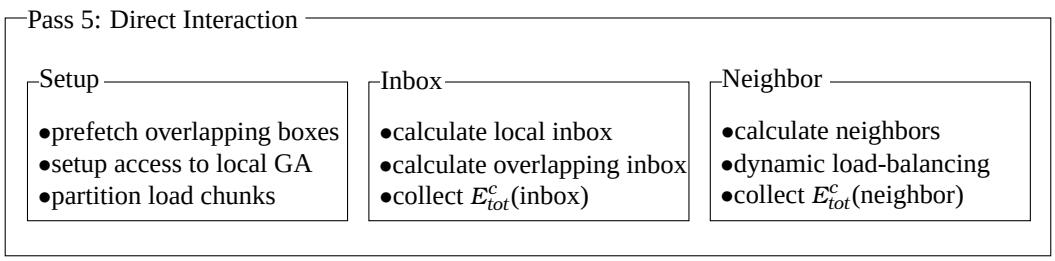


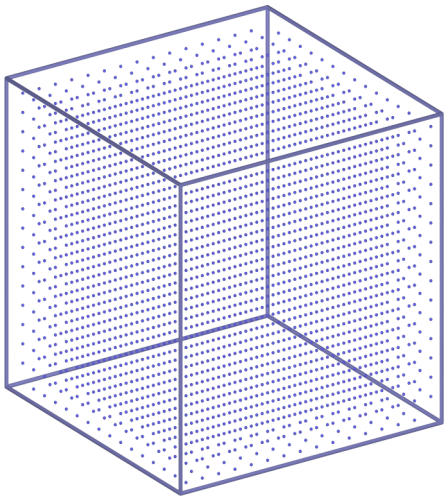

Figure 4.11: Parallel pass 5. Dynamic load balancing is applied to box neighbor-box interactions only.

| Method | Rate | Total Time |
|---|---|---|
| GA `get,put,acc` | 100% | 2.85 seconds |
| GA and local access | 86% | 2.45 seconds |

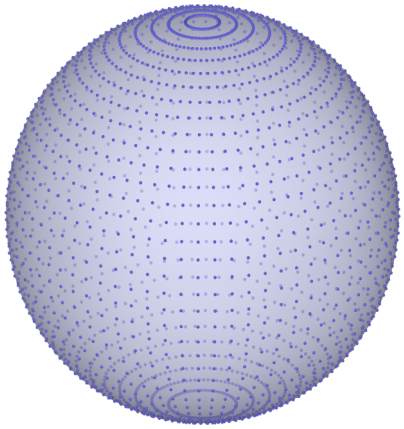
Table 4.5: Access to local memory can be performed faster, when memory is accessed directly without using GA get/put calls. (See Figure 4.10 for details) However, concurrent write operations are not prevented by the GA library. The programmer has to guarantee that no concurrent operations occur, otherwise the data in the accessed memory range may become inconsistent.

## 4.8 Load-Balancing

Load balancing is not necessary in a sequential program. A parallel program can suffer load imbalances and therefore cannot guaranty total balance in the first place. Efficiency depends on the data distribution and the communication time between processes. A distinction is drawn between static and dynamic load balancing. The first one operates on data distributed prior to computation. A subsequent distribution is not performed. Dynamic load balancing allows subsequent data distribution. Indeed, this is more expensive, but has the advantage that all processes are kept busy and idle time can be reduced. This implementation of the FMM handles both static and dynamic load balancing for pass 5.



(a) Homogeneous distribution



(b) Inhomogeneous distribution

Figure 4.12: Test cases: (a) Homogeneous particle distribution; particles reside inside the box on grid points. (b) Inhomogeneous particle distribution; particles reside only on the surface of the shaded sphere.

### 4.8.1 Static Load-Balancing

Parallelism in the FMM is achieved through distributed data. Each process executes the same code on different parts of the data. Thereby, the input data – coordinates and charges – are distributed evenly between the processes. Each process holds the same amount of particles independently from the particle distribution in the simulation box. However, the number of boxes containing the particles does not need to be equal for all processes. Good load balancing can be achieved for homogeneous distributions. However, static load balancing is not sufficient for inhomogeneous particle distributions as shown in Figure 4.12b. Some processes become idle and the efficiency decreases.

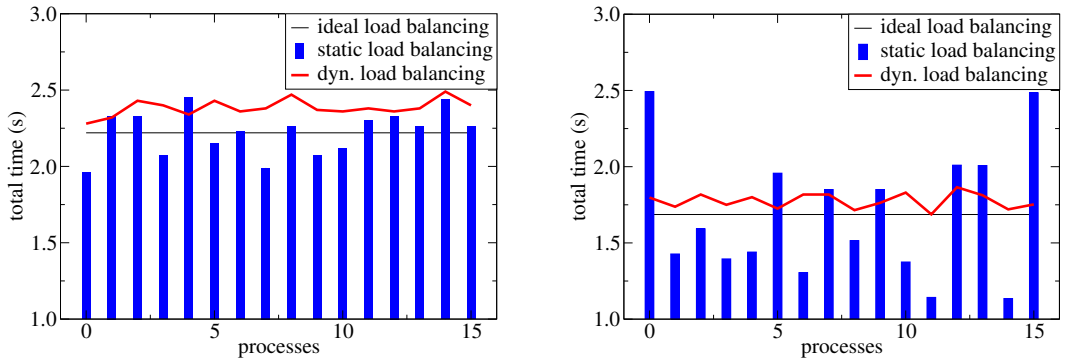
### 4.8.2 Dynamic Load-Balancing

Dynamic load balancing was implemented to be independent from the particle distribution in the simulation box and to avoid load imbalance. Thus, it is possible to reduce imbalance during the calculation, which was not possible in the static case. A process finished with its local data would become idle. Dynamic load balancing allows to transfer outstanding calculations from remote processes to another process which has already finished its local

work. This scheme is now outlined for pass 5.

In the beginning of pass 5, the data is distributed as shown for the static case. Firstly, all intra box interactions are calculated. Therefore, a dynamic load balancing scheme is not necessary since the static load balancing leads to adequate balance. This does not hold for box-box interactions. Imbalance is possible and must be reduced. The procedure is as follows. Each process subdivides its local boxes into a certain number of groups. These groups of boxes define the granularity of the code. After finishing the calculation of a group, a globally available progress counter is increased. The counter holds the current progress for each process. Thus, there are $p$ counters for $p$ processes organized in a global array. Incrementing the counter is done locally, thus the global array does not need to be synchronized for writing. Each process has to look up the local progress state only. Load balancing is activated by the first process finishing its local groups. This process becomes a support process, since there are no local calculations left. Now the supporting process can access the global array storing all progress informations. All processes still busy with their local data are not affected. After receiving the progress list, a host process with minimum progress is chosen to be supported. An additional global array allows to send a request to the remote host to offer support. Afterwards the support process waits for acknowledgement. It becomes necessary since the remote process still operates with direct data access. It is not possible, if two processes work on neighboring data. Especially concurrent writing in one and the same box could falsify the result. Allowing access to data via GA operations only, guarantees that concurrent data access does not occur. After receiving the acknowledgement from the remote host, the support process is able to pick up an unique remote group number with data. This data is copied into a local buffer and is processed. The host and support processes are involved in the load balancing scheme only. The remaining processes are not influenced. Moreover, this scheme can handle offers from more than one support process. The usage of a global counter for every process and lock operations permitting concurrent writing guarantee that one and the same group of particles is not processed twice.

To save computation time for homogeneous distributions and avoid gratuitous load balancing the support is limited to a certain group number. Thus, it would be more expensive to distribute the last groups to remote processes. Hence, the last groups of local data are processed locally. The number of groups available for support was determined experimentally and depends on the the communication time and bandwidth of the network.



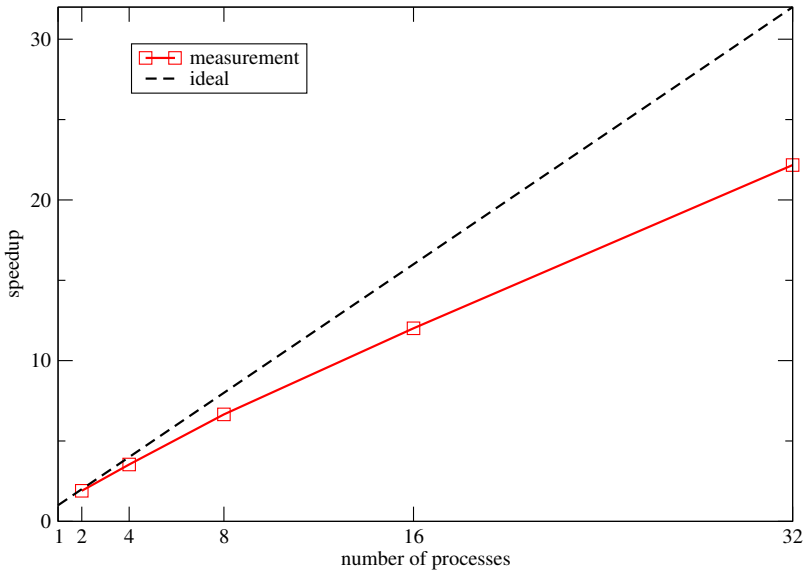

(a) sufficient static load balancing (bars)

(b) insufficient static load balancing (bars)

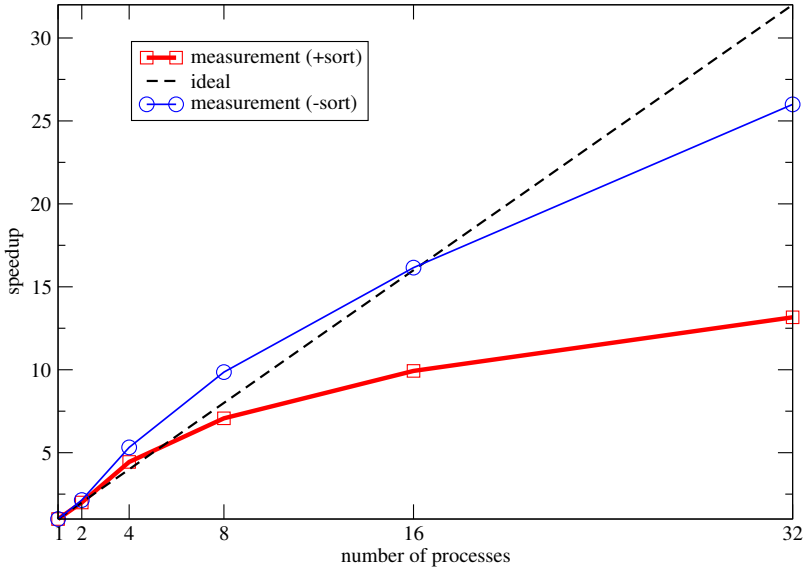
Figure 4.13: Static load balancing is adequate for homogeneous distributed particles. To assure parallel efficiency for inhomogeneous distributions a dynamic load balancing has to be implemented.

## 4.9 Parallel Scaling

A particle system with $8^7$ particles was chosen to determine the scaling. All particles are homogeneously distributed and reside on a grid. The maximal number of processes was chosen as 32. A processor number larger than 32 is not reasonable, since the calculation time for 32 processes amounts to only few seconds and the running time is already dominated by GA initialization routines. Scaling figures for pass 2 and pass 5 are shown in Figure 4.14, since only these passes were considered for parallelization. The discrepancy between the ideal scaling and the achieved scaling can be explained as follows. Increasing the number of processes leads to an increasing number of boxes at the edge of a processor's domain. Therefore, more boxes have to be transmitted to neighbor processes. At the same time, the number of local data accesses decreases. The limiting part of the scalability for pass 2 is the number of boxes at the edge of a processor's domain, and also sorting the particles into lowest level boxes. The scaling for pass 2 is shown with and without sorting. Superlinear behavior in the range from 2 to 8 processes is accomplished because of cache effects. The scaling for pass 2 can be seen as a worst case scenario, since it shows the initialization run for the FMM [2]. This run is completely identical to pass 2, except there is no calculation but only communication. All subsequent passes do not need to re-sort the data and have to calculate only the interactions.




(a) Pass 5 scaling plot




(b) Pass 2 scaling plot

Figure 4.14: (a) Parallel scaling for pass 5. (b) Parallel scaling for pass 2. The bold line represents the parallel scaling for the setup in pass 2. Succeeding runs of pass 2 can operate on pre-sorted data for each tree level. Thus, the thin line represents parallel scaling of pass 2 for pre-sorted data.

# 5 Summary and Outlook

This thesis covers the implementation of a gradient algorithm and the parallelization of the fast multipole method. In order to describe the implementation the FMM gradient, all necessary FMM fundamentals are introduced in the second chapter. The linear scaling of the sequential FMM, as well as a scheme to reduce the operator costs are shown.

The third chapter describes the gradient fundamentals and two different gradient algorithms. A standard gradient algorithm is discussed and an alternative gradient algorithm is introduced. The improved implementation of the FMM gradient offers two advantages compared to the standard implementation. Firstly, no complicated derivatives have to be calculated at runtime. All necessary derivatives can be acquired from the multipole expansion at the particle position and the shifted Taylor-like expansion. In contrast to the standard implementation, undefined expressions in the derivatives are avoided. Secondly, gradient computation time is reduced by 20% independently of the multipole length. The order of magnitude of the error in comparison to the standard gradient is near machine precision. Thus, the improved computation scheme does not introduce additional errors.

The fourth chapter describes the parallelization of the FMM. A parallel FMM algorithm was implemented using the *Global Arrays toolkit*, maintaining the sequential FMM data structures. Static and dynamic load balancing were implemented, since the parallelization of $N$-body algorithms, in general, creates an unstructured communication pattern. To achieve high performance for $N$-body simulations, load imbalances due to inhomogeneous particle distributions have to be reduced dynamically. The tradeoff between maintaining data locality and equally balanced computational load requires schemes that combine static load balancing schemes necessary for data locality with dynamic load balancing schemes that improve the parallel efficiency of the FMM. For performance reasons, the parallelization made use of maximal data locality. The granularity in the parallel data structures was chosen such that the decomposition and distribution of the data matches the inherent decomposition in the sequential FMM. The parallel performance was tested for particle systems containing 2 million and 16 million particles.

The parallel implementation shows good scaling properties for up to 32 processes. An even higher processor number would require larger particle systems, since the computation time per process for 2 million particles amounts to some seconds only. However, the implementation could be improved by merging remote boxes together prior to transmission. Memory is used efficiently, since no data in the magnitude of the input data is stored redundantly. As a side effect, the accuracy of the computation is improved with an increasing number of processes, because of the tree-like summation of the data. Indeed, while the required error bound was preserved in the sequential version, the parallel version reduced the actual computational error even further, as each process sums up its fraction of the total energy by itself, and afterwards computes the total energy by adding up all fractions from the different processes. This is similar to a tree-like adding scheme. Rounding errors are avoided, since most of the data has the same order of magnitude. The parallel version of the FMM can handle both homogeneous or inhomogeneous particle distributions efficiently using a dynamic load balancing scheme.

The parallel version of the FMM offers the possibility to calculate very large particle systems with billions of particles. The data distribution allows to use the available memory very efficiently. Also smaller systems will gain from the parallelization, since a computation time of only a few seconds enables molecular dynamic simulations to perform single time steps almost in real time.

# Bibliography


[1] S. Aluru and F. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *Proc. 4th International Conference on High-Performance Computing*, pages 230–235, 1997.

[2] D. H. Bailey. Highly parallel perspective: Twelve ways to fool the masses when giving performance results on parallel computers. *Supercomputing Review*, 4(8):54, August 1991.

[3] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.

[4] R. Beatson and L. Greengard. A short course on fast multipole methods. pages 1–37, 1997.

[5] J. A. Board, Jr., J. W. Causey, J. F. Leathrum, Jr., A. Windemuth, and K. Schulten. Accelerated molecular-dynamics simulation with the parallel fast multipole algorithm. *Chem. Phys. Lett.*, 198(1):89–94, October 1992.

[6] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. of Computation*, 31:333–390, 1977.

[7] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Computational Chemistry*, 4(2):187 – 217, 1983.

[8] D. R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[9] M. S. Gordon C. H. Choi, K. Ruedenberg. New parallel optimal-parameter fast multipole method (OPFMM). *J. Comp. Chem.*, 22:1484–1501, 2001.

[10] C. H. Choi, J. Ivanic, M. S. Gordon, and K. Ruedenberg. Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion. *The Journal of Chemical Physics*, 111(19):8825–8831, 1999.

[11] T. Darden, D. York, and L. Pedersen. Particle mesh Ewald — an $N \log N$ method for Ewald sums in large systems. *J. Chem. Phys*, 98:10089–10092, June 1993.

[12] Junichiro Makino Department. Grape project.

[13] H. Q. Ding, N. Karasawa, and W. A. Goddard, III. Atomic level simulations on a million particles: the cell multipole method of Coulomb and London nonbond interactions. *J. Chem. Phys.*, 97(6):4309–4315, September 1992.

[14] J. Dongarra and F. Sullivan. Guest Editors' Introduction: The Top 10 Algorithms. *Computing in Science and Engg.*, 2(1):22–23, 2000.

[15] P. P. Ewald. The calculation of optical and electrostatic grid potential. *Ann. Phys.*, 64:253, 1921.

[16] A. Geist, A. Beguelin, J. Dongarra, We. Jiang, R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Scientific and engineering computation. 1994.

[17] L. Greengard. The numerical solution of the n-body problem. *Comput. Phys.*, 4(2):142–152, 1990.

[18] L. Greengard and W. Gropp. A parallel version of the fast multipole method-invited talk. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 213–222, Philadelphia, PA, USA, 1989. Society for Industrial and Applied Mathematics.

[19] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[20] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, William Saphir, , and M. Snir. *MPI — The Complete Reference: Volume 2, the MPI-2 Extensions*. MIT Press, 1998.

[21] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. 1988.

[22] S. Hoefler-Thierfeldt. *User Documentation. Juelich Multiprocessor JUMP*, 2004.

[23] M. Hofmann. Paralleles Sortieren am Beispiel der schnellen Multipolmethode. Master's thesis, Chemnitz University of Technology, 2005.

[24] C. L. Brooks III, B. M. Pettitt, and M. Karplus. Structural and energetic effects of truncating long ranged interactions in ionic and polar fluids. *The Journal of Chemical Physics*, 83(11):5897–5908, 1985.

[25] Shimada J, H. Kaneko, and T. Takada. Performance of fast multipole methods for calculating electrostatic interactions in biomacromolecular simulations. *J. Comput. Chem.*, 15(1):28–43, 1994.

[26] A. E. Koniges, editor. *Industrial Strength Parallel Computing*. 2000.

[27] M. F. Mokbel, W. G. Aref, and I. Kamel. Analysis of multi-dimensional space-filling curves. *Geoinformatica*, 7(3):179–209, 2003.

[28] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global Arrays: A nonuniform memory access programming model for high-performance computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.

[29] J. Nieplocha, Jialin Ju, M. K. Krishnan, B. Palmer, and Vinod Tipparaju. The Global Arrays user's manual. Technical report, Pacific Northwest National Laboratory, 2002.

[30] L. S. Nyland, J. F. Prins, and J. H. Reif. A data-parallel implementation of the adaptive fast multipole algorithm. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 111–123, Hanover, NH, 1993.

[31] H. G. Petersen, D. Soelvason, J. W. Perram, and E. R. Smith. The very fast multipole method. *J. Chem. Phys.*, 101(10):8870–8876, November 1994.

[32] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and Brian P. Flannery. *Numerical Recipes in FORTRAN; The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1993.

[33] T. Rauber and G. Rünger. *Parallele und verteilte Programmierung – Architektur, Programmierung, Algorithmen*. Springer, 2000.

[34] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *J. Comp. Phys.*, 60:187–207, October 1985.

[35] L. De Rose. The hardware performance monitor toolkit. In *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, pages 122–131, London, UK, 2001. Springer-Verlag.

[36] K. E. Schmidt and M. A. Lee. Implementing the fast multipole method in three dimensions. *J. Stat. Phys.*, 63:1223–1235, 1991.

[37] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87(1–2):266–290, 1995.

[38] C. A. White and M. Head-Gordon. Derivation and efficient implementation of the fast multipole method. *J. Chem. Phys*, 101:6593–6605, October 1994.

[39] C. A. White and M. Head-Gordon. Fractional tiers in fast multipole method calculations. *J. Chem. Phys. Lett.*, 257(5):647, 1996.

[40] C. A. White and M. Head-Gordon. Rotating around the quartic angular momentum barrier in fast multipole method calculations. *J. Chem. Phys*, 105:5061–5067, September 1996.

[41] B. Wilkinson and M. Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

# A Computational Resources

All computations were performed on JUMP [22]. Computations with less than 32 processors are guaranteed to run on a single frame .

## A.1 Hardware

SMP Cluster

### IBM p690 Frame Characteristics

- 32 processors, Power4+, 1.7 GHz
- Main Memory: 128 GB, 567 MHz
- Internal L1 cache: 64 KB instruction, 32 KB data (per processor)
- Shared L2 cache: 1.5 MB (per chip = 2 processors)
- Shared L3 cache: 512 MB (per frame)
- Peak performance: 218 GFLOPS

### Cluster Characteristics

- Total number of p690 frames: 41
- Total number of processors: 1312
- Aggregate peak performance: 8.9 TFLOPS
- LINPACK performance (41 nodes): 5.568 TFLOPS
- Aggregate main memory: 5.2 TByte

- Global disk space (GPFS): 8 x 7 x 14 x 72 GB = 56 TB
- Cluster interconnect: HPS - High Performance Switch:
  - Bandwidth > 1400 MB/s per link
  - Latency < 6.5 us

## A.2 Software

### Operating System

IBM AIX 5.2

### Compiler

IBM XL Fortran Enterprise Edition, version 9.1
IBM XL C/C++ Enterprise Edition, version 7.0

### Global Arrays

Version 3.3
Compiler switches:

```
USE_MPI = y
TARGET = IBM64
```