Development of a Massively Parallelized Micromagnetic Simulator on Graphics Processors

A thesis submitted for the degree of
Master of Science

by

Aamir Ahmed Khan
Institute of Nanoelectronics
Technische Universität München

October 14, 2010

Thesis Advisors

Prof. György Csaba, University of Notre Dame
Prof. Wolfgang Porod, University of Notre Dame
Prof. Paolo Lugli, Technische Universität München

Dedicated to
my beloved parents

without their love and support, nothing could have been possible.

Acknowledgements

I want to deeply thank my advisor Prof. György Csaba, who offered me the opportunity to do this thesis under his supervision. He had been very kind to me throughout the thesis and gave his valuable advice and mentoring. Most of the ideas in this thesis are due to his guidance and suggestions. I also want to thank Prof. Wolfgang Porod, who appointed me as a visiting researcher in his group at the University of Notre Dame and provided financial and logistical support for this work. I am also highly obliged to Prof. Paolo Lugli who served as my official advisor at Technische Universität München and took keen interest in my research.

I also cannot forget to mention Heidi Deethardt, Edit Varga, Mirakmal Niyazmatov and Anand Kumar for helping me throughout my stay at Notre Dame.

Abstract

This thesis describes the development of a high performance micromagnetic simulator running on massively parallel Graphics Processing Units (GPU's). We show how all components of the simulator can be implemented from scratch. Special attention is given to the calculation of magnetic field to reduce its running time, which traditionally takes $O(N^2)$. This is also the most computationally intensive part of the simulator and we have shown its GPU implementation as well. We also have used two different algorithms for field calculation – (1) the traditional direct summation and (2) the fast multipole method (FMM). Direct summation is better suited for small problems, while large problems are best handled by FMM due to its $O(N \log N)$ complexity. When compared to the single-threaded implementation on CPU, performance benchmarks show a maximum speed-up in excess of 30x while running on GPU.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Nanomagnetism is a flourishing research field – it generated interest both in the scientific and the engineering community [1]. Nanomagnets show strong size dependent behaviors and ordering phenomena, which is interesting for basic physics. Introducing nanomagnets into silicon chips (Fig. 1.1) may revolutionize microelectronics [2] [3].

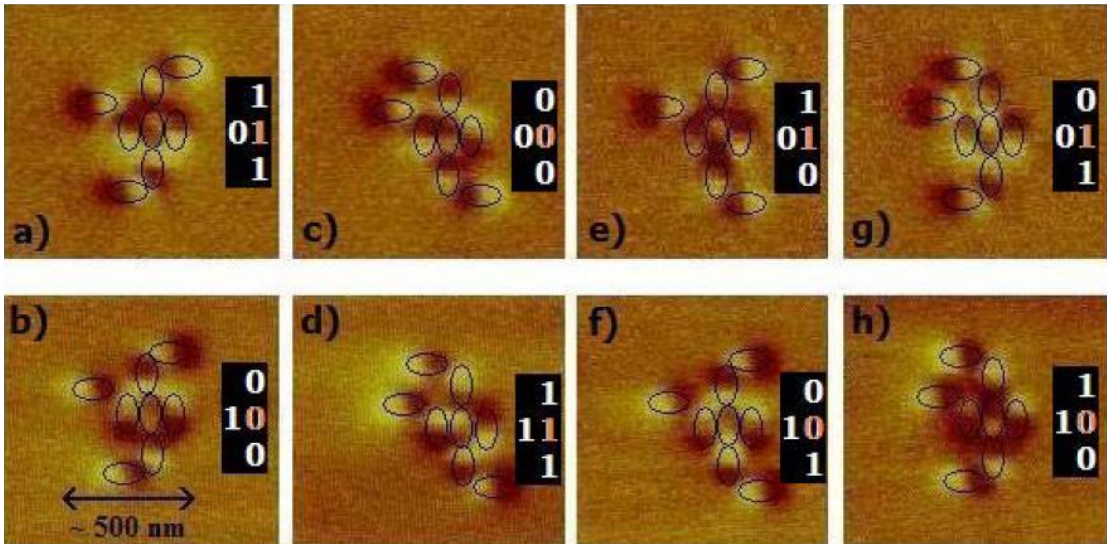


Figure 1.1: A nanomagnetic logic majority gate [2].

Ferromagnetic behavior of large magnets is usually described by hysteresis behavior, which is the signature of the average behavior of a large number of magnetic domains. If magnets are made smaller and their size becomes comparable to the natural domain size

(typically this happens if the magnets are around few hundred nanometers in size) then the domain patterns must exactly be calculated in order to understand magnetic behaviors. This can be done by applying micromagnetic theory.

The central task of micromagnetics is to calculate the dynamics and the steady states of the $\mathbf{M}(\mathbf{r}, t)$ magnetization distribution. The dynamics itself is described by the Landau-Lifshitz equation and this equation contains all magnetic fields generated by the sample, the external field and magnetic fields that account for quantum mechanical phenomena. Micromagnetic theory ignores quantum mechanics but it is proven to be working very well on samples larger than a few nanometers in size.

Microelectronic calculations are very time consuming. A magnetic sample can be several micrometers large, but to catch the details of domain patterns, it has to be discretized into very small (few nm) size mesh elements. The Landau-Lifshitz equation is a strongly nonlinear, time-dependent partial differential equation (PDE) and to solve it at each iteration step, one has to recalculate the distribution of all magnetic fields. On a desktop PC one can run simulation on a few micron size magnets. Simulating few tens of micron size structures is possible but takes weeks and practically unusable for any engineering purpose. Using supercomputers does not accelerate calculations significantly, since the calculation is difficult to decompose into independent domains for the solution.

1.2 Motivation

General Purpose Processing on Graphics Processing Units (GPGPU) is a recent development in computing. Graphics processors (GPU's) were around for a long time, but were used almost exclusively to speed up graphics rendering of demanding computer games. The kind of instructions they perform are quite limited, but a large number of processors work on the same task (i.e. rendering a particular part of the image). In many senses, rendering is very similar to the basic, repetitive computations that is required in most of the scientific computations. Graphics card manufacturers (among them NVIDIA probably the first time) realized the potential of their hardware for scientific purposes and developed tools to ease general purpose programming on their hardware [4].

GPU accelerated computing is a flourishing area of scientific computing. Part of its success is the excellent price/performance ratio on GPU chips: for around $2000, one can buy 512 computing cores, which is orders of magnitude better deal than multi-core CPUs. The cores can communicate with each other with extremely high-speed internal channels,

Figure 1.2: NVIDIA Tesla GPU.

which makes it possible to parallelize applications with a lot of communication between computational domains.

The above arguments make clear the motivation to develop a GPU-accelerated micromagnetic simulator. This thesis describes our effort to do that.

1.3 Overview of the Work

We decided to build our simulator *from scratch*, since it was difficult to understand and paralellize a code that was not developed in-house. The next two chapters of this thesis describe the micromagnetic background required to build this simulator.

The most time consuming part of a micromagnetic calculation is to determine the magnetic field (or magnetic scalar potential) that a sample generates. Most of our thesis is devoted to this problem. First we show a direct, brute force approach for the calculation and then a very complicated, state of art method – the fast multipole method (FMM) in chapter 4.

The dynamics of a micromagnetic sample are governed by the Landau-Lifshitz equations. We describe how to build a solver to solve a system of millions of these differential equations in chapter 5.

We then show sample problems that demonstrate our code working correctly (chapter 6). We parallelize the code for both the direct method and the FMM and benchmark the speed-ups (chapter 7). We demonstrate that for simple problems, the direct method is

quite efficient, since it is easy to paralellize and carries almost no overhead. For larger size problems (mesh elements $> 10^5$), the FMM-based methods overtakes the direct calculation.

Only very recently one other group demonstrated results on GPU-accelerated solvers, based on similar principles [5]. We believe that our work is one of the first attempts to realize a high-performance, GPU-based complete micromagnetic simulator. Although the code is still unoptimized, the potential of a scalable, massively parallelized code is already apparent. We hope it will become a design tool for future nanomagnetic computing devices.

Chapter 2

Fundamentals of Micromagnetic Simulations

2.1 Magnetization

We discretize the 3-dimensional magnetic material into a computational mesh of tiny mesh elements with dimensions $\Delta x, \Delta y, \Delta z$. Each mesh element at position $\mathbf{r}$ is represented by a magnetic moment called magnetization $\mathbf{M}(\mathbf{r})$. The moments originate from the electron spins of respective atoms in the material and their collective behaviour is what makes the material ferromagnetic, diamagnetic or paramagnetic. All of these moments have same magnitude, termed as saturation magnetization (M_s) of the material, but differing only in their orientation (Fig. 2.1). In micromagnetics, we are interested in the steady state as well as transient behavior of individual moments in response to different magnetic fields.

Figure 2.1: Two different magnetization distributions.

2.2 Landau-Lifshitz-Gilbert Equation

The equation that governs the dynamics of magnetization distribution in a magnetic material is due to Lev Landau, and Evgeny Lifshitz and T. L. Gilbert and is called Landau-Lifshitz-Gilbert (LLG) equation.

$$\frac{\partial}{\partial t}\mathbf{M}(\mathbf{r},t) = -\gamma \mathbf{M}(\mathbf{r},t) \times \mathbf{H}_{\text{eff}}(\mathbf{r},t) - \frac{\alpha\gamma}{M_s}\mathbf{M}(\mathbf{r},t) \times \Big(\mathbf{M}(\mathbf{r},t) \times \mathbf{H}_{\text{eff}}(\mathbf{r},t)\Big) \tag{2.1}$$

Here γ is the electron gyromagnetic ratio, α is the damping constant and $\mathbf{H}_{\text{eff}}$ is the effective magnetic field acting on the magnetization. The effect of LLG equation is to align or anti-align the magnetization vector with the effective field by going through a precession motion as shown in Fig. 2.2.

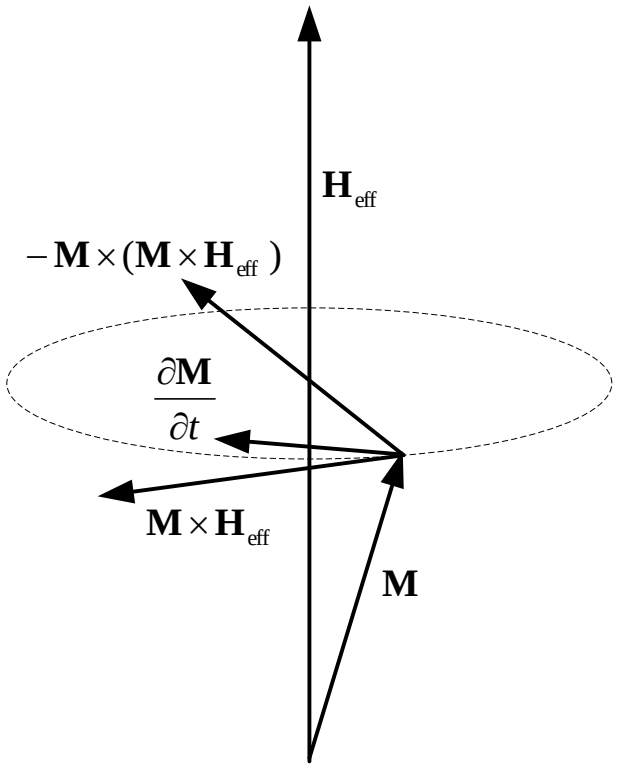


Figure 2.2: Landau-Lifshitz-Gilbert Equation.

2.3 Effective Field

The goal of a micromagnetic simulation is to compute the time-evolution of an $\mathbf{M}(\mathbf{r}, t)$ magnetization distribution by solving Landau-Lifshitz equation. The change in the magnetization is due to the $\mathbf{M} \times \mathbf{H}_{\text{eff}}$ torque generated by the effective magnetic field, which is described as

$$\mathbf{H}_{\text{eff}} = \mathbf{H}_{\text{exch}} + \mathbf{H}_{\text{demag}} + \mathbf{H}_{\text{ext}} + \mathbf{H}_{\text{therm}} + ... \tag{2.2}$$

Any other physical phenomenon can be incorporated into the simulation by adding the respective magentic field component to the effective field. Calculation of this effective field is the most essential component of any micromagnetic simulation. In the following sections, the most important and frequently used components of effective fields are described.

2.3.1 Exchange Field

Exchange field represents the quantum-mechanical forces among the magnetization vectors in a quasi-classical way. It is responsible for aligning the magnetization vectors with each other and is defined as [1],

$$\mathbf{H}_{\text{exch}}(\mathbf{r}) = \frac{2A_{\text{exch}}}{\mu_0 M_s^2} \nabla^2 \mathbf{M}(\mathbf{r}), \tag{2.3}$$

where μ_0 is the magnetic permeability of free space, A_{exch} is a material parameter called the exchange stiffness and ∇^2 is the vector Laplacian operator. In the case of cubic discretization of computational mesh ($\Delta x = \Delta y = \Delta z$), it can also be calculated as the sum of neighboring magnetizations[1].

$$\mathbf{H}_{\text{exch}}(\mathbf{r}) = \frac{2A_{\text{exch}}}{\mu_0 M_s^2} \frac{1}{\Delta x^2} \sum_{\mathbf{r}_i \in \text{neighbors}} \mathbf{M}(\mathbf{r}_i) \tag{2.4}$$

However in both the equations, the calculation of exchange interaction remains local, $O(N)$, since it only depends upon the magnetizations in its immediate neighborhood.

2.3.2 External Field

This component $\mathbf{H}_{\text{ext}}$ represents any external magnetic field to which the magnetization distribution is exposed to. It is just an additive component to the effective field and is not

[1] The exchange field is not uniquely defined: for example one can add any vector parallel to $\mathbf{M}$ without changing the torque in Eq. (2.1)

a function of any neighboring magnetization vectors, thus also scales with $O(N)$.

2.3.3 Demagnetization Field

The demagnetization field, also called as magnetostatic field, describes the long-range magnetic field created by the $\mathbf{M}(\mathbf{r})$ distribution. The evaluation of demagnetization field is most time consuming part of the calculation since it represents all the pairwise interactions between magnetization vectors and requires $O(N^2)$ steps. As already discussed, all the other steps are local and scale with $O(N)$. The calculation of demagnetization field is so involved that it is treated in a separate chapter (Ch. 3).

2.4 Components of a Micromagnetic Simulator

There are a number of publications describing the construction of a micromagnetic simulator [1]. Fig. 2.3 summarizes the main steps of the simulation flow.

We start with the magnetization state $\mathbf{M}(\mathbf{r}, t)$ and use it to calculate the demagnetization and exchange fields, which are then added to any external field and other components to get the effective field $\mathbf{H}_{\text{eff}}$. The effective field is then used to calculate $\mathbf{M} \times \mathbf{H}_{\text{eff}}$ torque on each magnetization vector and the new magnetization state $\mathbf{M}(\mathbf{r}, t + \Delta t)$ according to the LLG equation. The above steps are then repeated for each time-step until required or there is not sufficient dynamic activity in the magnetization state.

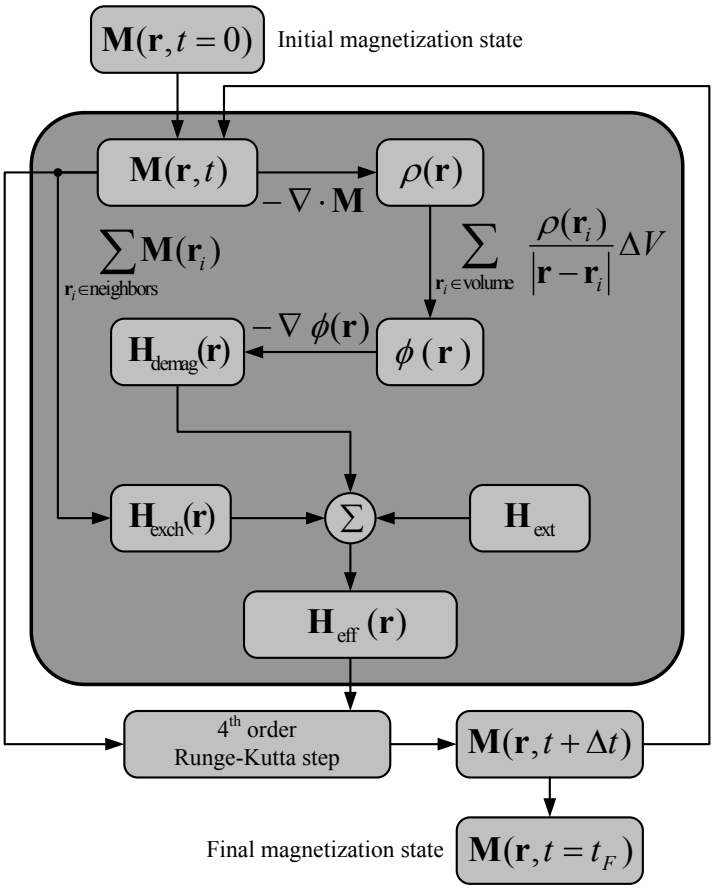


Figure 2.3: Flow chart of a micromagnetic simulation.

Chapter 3

Magnetostatic Field Calculation

Calculation of effective field is the most essential component of a micromagnetic simulation, and the component of effective field (Eq. (2.2)) which is the most time consuming to calculate is magnetostatic field, also called as demagnetizing field. This is because this field describes the long-range magnetostatic field created by the magnetization distribution and that it has to be calculated by considering all pairwise interactions among all the mesh points in the computational domain.

Our goal is to calculate magnetostatic field, $\mathbf{H}_{\text{demag}}(\mathbf{r})$, from the given magnetization distribution, $\mathbf{M}(\mathbf{r})$. The magnetization is given on a 3-dimensional computational mesh for each discretized mesh element of dimensions $\Delta x, \Delta y, \Delta z$.

In this chapter, we describe two schemes for the calculcation of magnetostatic field.

3.1 Field from Magnetization

The field is calculated in two steps. First we calculate the magnetic flux density ($\mathbf{B}$) from the individual magnetization vectors by summing them up over the entire volume.

$$\mathbf{B}(\mathbf{r}) = \frac{\mu_0}{4\pi} \sum_{\mathbf{r}_i \in \text{volume}} \left(\frac{3\left\{\mathbf{M}(\mathbf{r}_i) \cdot (\mathbf{r} - \mathbf{r}_i)\right\}(\mathbf{r} - \mathbf{r}_i)}{|\mathbf{r} - \mathbf{r}_i|^5} - \frac{\mathbf{M}(\mathbf{r}_i)}{|\mathbf{r} - \mathbf{r}_i|^3} \right) \Delta x \Delta y \Delta z \qquad (3.1)$$

From the flux density, the magnetostatic field can be calculated using the straightfor-

ward relation.

$$\mathbf{B} = \mu_0(\mathbf{H}_{\text{demag}} + \mathbf{M}) \tag{3.2}$$

$$\mathbf{H}_{\text{demag}} = \frac{\mathbf{B}}{\mu_0} - \mathbf{M} \tag{3.3}$$

Fig. 3.1 and Fig. 3.2 show two examples of calculating field using this approach.

3.2 Field from Magnetic Scalar Potential

There is another method for the calculation of magnetostatic field which requires the calculation of magnetic scalar potential first. This method is longer but has certain advantages which will be discussed in section 3.3. The method is outlined in Fig. 3.3 and 3.4 and is discussed below.

3.2.1 Magnetic Charges

The calculation of magnetostatic field starts by calculating the magnetic volume charge density over the entire mesh using the divergence operation [6].

$$\rho(\mathbf{r}) = -\nabla \cdot \mathbf{M}(\mathbf{r}) \tag{3.4}$$

$$= -\left(\frac{\partial M_x}{\partial x} + \frac{\partial M_y}{\partial y} + \frac{\partial M_z}{\partial z}\right) \tag{3.5}$$

The finite difference counterpart of this operation is given as

$$\begin{aligned} \rho(\mathbf{r}) = & - \frac{M_x(x+\Delta x, y, z) - M_x(x-\Delta x, y, z)}{2\Delta x} \\ & - \frac{M_y(x, y+\Delta y, z) - M_y(x, y-\Delta y, z)}{2\Delta y} \\ & - \frac{M_z(x, y, z+\Delta z) - M_z(x, y, z-\Delta z)}{2\Delta z} \end{aligned} \tag{3.6}$$

For magnetic field calculations, one generally has to take into account not only volume charges but also surface charges, which appear at the boundaries of magnetic bodies and where Eq. (3.6) cannot be interpreted due to the discontinuity of the magnetization. To keep the algorithm simple, we do not explicitly consider surface charges, but instead we interpret Eq. (3.6) over the entire volume. This 'smears' an equivalent volume charge on

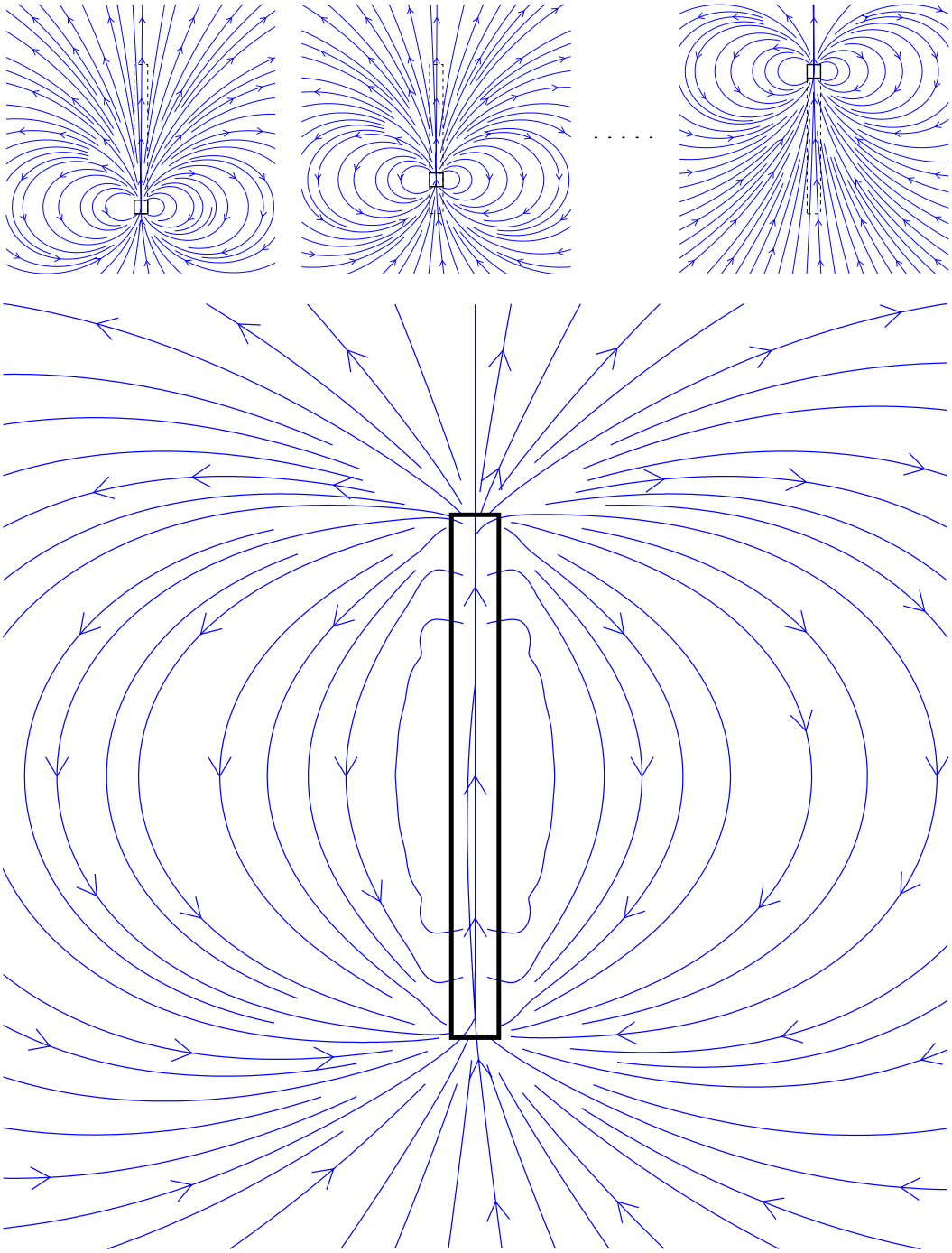
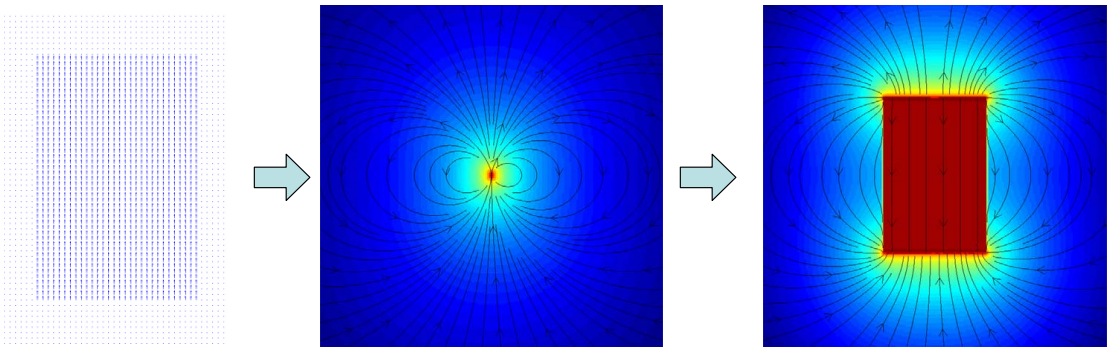


Figure 3.1: Summation of magnetic flux density from individual magnetizations.



(a) (b) (c)

Figure 3.2: Summation of magnetostatic field of a bar magnet from individual magnetizations. (a) Magnetization distribution $\mathbf{M}$. (b) Flux density $\mathbf{B}$ due a single magnetization vector. (c) Magnetostatic field $\mathbf{H}_{\text{demag}}$.

the cells adjacent to the surface. There is some error in the magnetic field at the boundaries of magnetized regions, but this error becomes negligible on the neighboring mesh points.

3.2.2 Magnetic Scalar Potential

From the magnetic charges, we evaluate the magnetic scalar potential at each mesh point using the follwing expression, analogous to Coulombic and Newtonian $\frac{1}{r}$ potential.

$$\Phi(\mathbf{r}) = \frac{1}{4\pi} \sum_{\mathbf{r}_i \in \text{volume}} \frac{\rho(\mathbf{r}_i)}{|\mathbf{r} - \mathbf{r}_i|} \cdot \Delta x \Delta y \Delta z \tag{3.7}$$

3.2.3 Magnetostatic Field

Once we have the potential, we can calculate the field as its gradient.

$$\mathbf{H}_{\text{demag}}(\mathbf{r}) = -\nabla \Phi(\mathbf{r}) \tag{3.8}$$

$$= \left[-\frac{\partial \Phi}{\partial x}, -\frac{\partial \Phi}{\partial y}, -\frac{\partial \Phi}{\partial z} \right] \tag{3.9}$$

The finite difference counterpart of this operation is given as

$$\begin{aligned} H_{\text{demag},\,x}(\mathbf{r}) &= -\frac{\Phi(x+\Delta x, y, z) - \Phi(x-\Delta x, y, z)}{2\Delta x} \\ H_{\text{demag},\,y}(\mathbf{r}) &= -\frac{\Phi(x, y+\Delta y, z) - \Phi(x, y-\Delta y, z)}{2\Delta y} \\ H_{\text{demag},\,z}(\mathbf{r}) &= -\frac{\Phi(x, y, z+\Delta z) - \Phi(x, y, z-\Delta z)}{2\Delta z} \end{aligned} \tag{3.10}$$

The finite difference calculation of divergence (Eq. (3.6)) and gradient (Eq. (3.10)) requires that there be at least one padding layer around the magnetization distribution (i.e. cells where $\mathbf{M} = 0$). This does not pose any serious problem however.

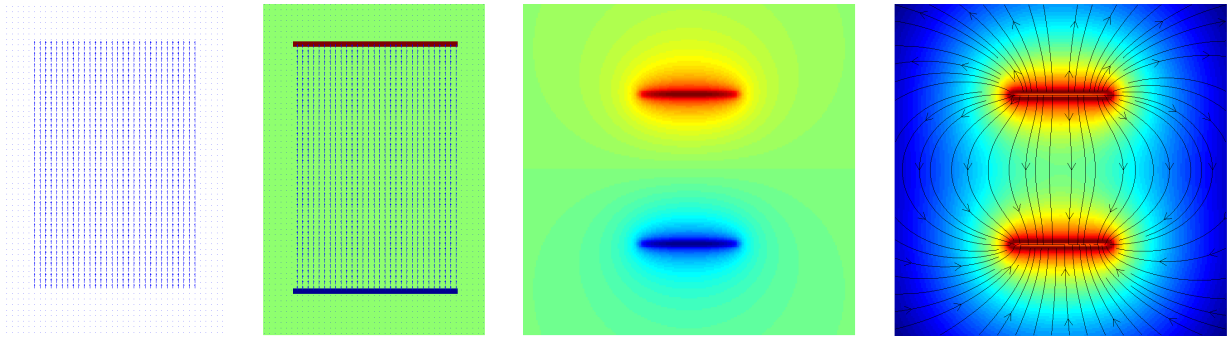


Figure 3.3: Magnetostatic field of a bar magnet through magnetic scalar potential. (a) Magnetization distribution $\mathbf{M}$. (b) Magnetic volume charge density ρ. (c) Magnetic scalar potential Φ. (c) Magnetostatic field $\mathbf{H}_{\text{demag}}$.

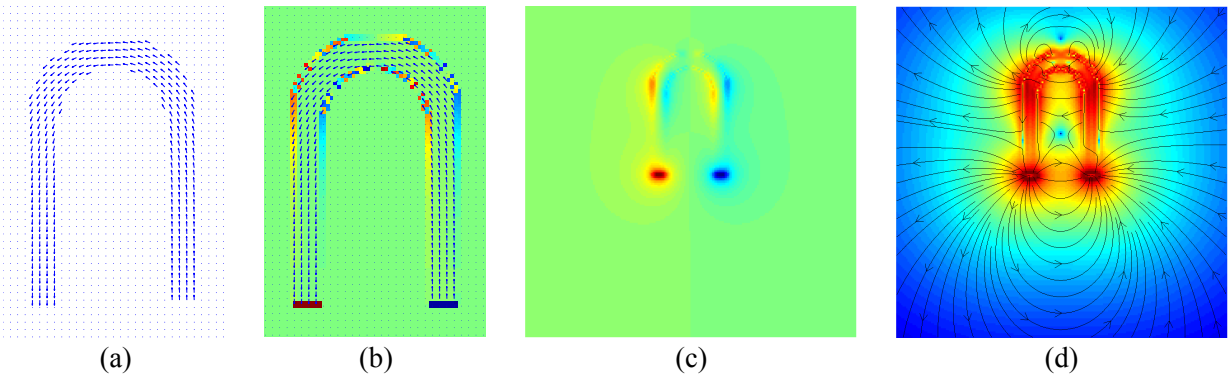


Figure 3.4: Magnetostatic field of a U-shaped magnet through magnetic scalar potential. (a) Magnetization distribution $\mathbf{M}$. (b) Magnetic volume charge density ρ. (c) Magnetic scalar potential Φ. (c) Magnetostatic field $\mathbf{H}_{\text{demag}}$.

3.3 Complexity of Field Calculation

Clearly, the most time consuming step of field calculation is the evaluation of Eq. (3.1) for magnetization method or Eq. (3.7) for the potential method. It represents all the pairwise interactions between magnetization vectors and requires $O(N^2)$ steps. Except it, all the other calculations are local and scale with $O(N)$.

A complexity of $O(N^2)$ is prohibitively huge for large-sized problems. We definitely need a fast method for this step to keep the simulation running time within reasonable limits for any appreciable problem size. The advantage of calculating field through the longer route of scalar potential is that potential theory is a well-established theory in mathematical physics. Several fast methods of calculating pairwise interactions have been devised such as Fast Fourier Transform (FFT), Fast Multipole Method (FMM), etc. These fast methods reduce the complexity of from $O(N^2)$ to $O(N \log N)$.

We preferred FMM over FFT because it can further reduce the complexity to $O(N)$. Also unlike FFT, FMM does not force us to use periodic boundary conditions. Periodic boundary conditions are generally undesirable because they necessitate the use of large padding areas around the physical structure to be simulated. Although FMM is an approximate method of potential calculation, it does not hurt us because we are anyway doing numerical simulations and do not require further accuracy than what is afforded by floating point standards. FMM is described in detail in chapter 4.

Chapter 4

Fast Multipole Method

Fast multipole method (FMM) [7] is a powerful mathematical technique to efficiently calculate pairwise interactions in physical systems, matrix-vector multiplication and summation [8]. It can do so in $O(N \log N)$ or even better $O(N)$ time instead of traditional $O(N^2)$ while sacrificing some accuracy. It is regarded as among the top-10 algorithms of the 20th century [9] and has already been exploited for micromagnetic simulations [10] [5].

4.1 Mathematical Description

In the context of potential theory, FMM enables to efficiently perform the summation of Eq. (3.7). The underlying idea is that instead of evaluating all pairwise interactions which would require $O(N^2)$ steps, the charges can be lumped together and the far field of this cluster at any far-lying point in space can be *approximated* with some acceptable accuracy, regardless of the distribution of charges in the cluster.

4.1.1 Harmonic functions

FMM is applicable to a particular class of fucntions called harmonics functions. These are the solution of Laplace equation

$$\nabla^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = 0 \tag{4.1}$$

For Coulombic and Newtonian potentials, the corresponding harmonic functions that

satisfies Eq. (4.1) are of the form

$$\Phi = \frac{q}{\sqrt{x^2 + y^2 + z^2}} \tag{4.2}$$

4.1.2 Mutipole Series for a Single Charge

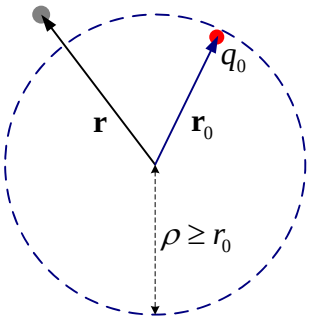


Figure 4.1: Single charge and single potential target.

To derive a series expansion for Eq. (4.2), we have to resort to spherical coordinates. Consider a charge q_0 at point $\mathbf{r}_0(r_0, \theta_0, \phi_0)$. The potential from this charge at any point $\mathbf{r}(r, \theta, \phi)$ is given by

$$\Phi(\mathbf{r}) = \frac{q_0}{|r - r_0|} \tag{4.3}$$

$$\Phi(\mathbf{r}) = \frac{q_0}{\sqrt{r^2 + r_0^2 - 2rr_0\cos\gamma}} \tag{4.4}$$

We are only interested to form a series expansion in the region $|\mathbf{r}| > |\mathbf{r}_0|$, so we can factor out r in the denominator

$$\Phi(\mathbf{r}) = \frac{q_0}{r\sqrt{2 + \left(\frac{r_0}{r}\right)^2 - 2\left(\frac{r_0}{r}\right)\cos\gamma}}, \tag{4.5}$$

where γ is the angle between $\mathbf{r}$ and $\mathbf{r}_0$. The radicand in the above equation is an infinite series in $\frac{r_0}{r}$ with Legendre functions, $P_l(\cos\gamma)$, as its coefficients [11].

$$\Phi(\mathbf{r}) = \frac{q_0}{r}\sum_{l=0}^{\infty} P_l(\cos\gamma)\left(\frac{r_0}{r}\right)^l \tag{4.6}$$

$$\Phi(\mathbf{r}) = q_0\sum_{l=0}^{\infty} P_l(\cos\gamma)\frac{r_0^l}{r^{l+1}} \tag{4.7}$$

The above equation contains γ which depends upon both $\mathbf{r}$ and $\mathbf{r}_0$. Our goal is to factor out these dependencies. Towards that end, we need a result from classical mathematics, called the addition theorem for spherical harmonics [11].

$$P_l(\cos\gamma) = \sum_{m=-l}^{l} Y_l^m(\theta,\phi) Y_l^{m*}(\theta_0,\phi_0), \tag{4.8}$$

where $Y_l^m(\theta,\phi)$ are the spherical harmonics described in section 4.1.5. Eq. (4.7) then takes the form of a multipole series.

$$\Phi(\mathbf{r}) = q_0 \sum_{l=0}^{\infty} \sum_{m=-l}^{l} Y_l^m(\theta,\phi) Y_l^{m*}(\theta_0,\phi_0) \frac{r_0^l}{r^{l+1}} \tag{4.9}$$

$$\Phi(\mathbf{r}) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \frac{M_l^m}{r^{l+1}} Y_l^m(\theta,\phi), \tag{4.10}$$

where M_l^m are the multipole coefficients defined as

$$M_l^m = q_0 r_0^l Y_l^{m*}(\theta_0,\phi_0) \tag{4.11}$$

Eq. (4.10), the multipole series, has now separate factors depending upon $\mathbf{r}(r,\theta,\phi)$ and $\mathbf{r}_0(r_0,\theta_0,\phi_0)$, which is the mathematical basis for fast multipole method.

4.1.3 Mutipole Series for a Charge Distribution

The multipole coefficients can further be extended to represent distribution of charges instead of just a single charge. Suppose we have K chages distributed inside a sphere of radius ρ as shown in Fig. 4.2. Then Eq. (4.10) still represents the potential at any point in the region $|\mathbf{r}| > \rho$ but with general multipole coefficients

$$M_l^m = \sum_{i=1}^{K} q_i r_i^l Y_l^{m*}(\theta_i,\phi_i) \tag{4.12}$$

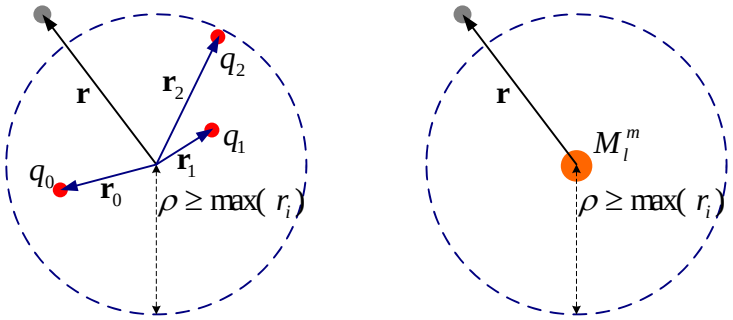


Figure 4.2: Multipole coefficients for charge distribution.

4.1.4 Approximation of Mutipole Series

For the multipole coefficients described by Eq. (4.12) and multipole series of Eq. (4.10), we can truncate the series at $l = P$, for an error bound of

$$\left| \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \frac{M_l^m}{r^{l+1}} Y_l^m(\theta,\phi) - \sum_{l=0}^{P} \sum_{m=-l}^{l} \frac{M_l^m}{r^{l+1}} Y_l^m(\theta,\phi) \right| = \epsilon \leq \frac{\sum_{i=1}^{K} |q_i|}{r-\rho} \left(\frac{\rho}{r}\right)^{P+1} \tag{4.13}$$

It can be clearly seen that as we move to points farther away from the charge distribution ($r > \rho$), the error falls rapidly. The rate of error decay w.r.t. P is given by [7].

$$\frac{d\epsilon}{dP} \approx \left(\sqrt{3}\right)^{-P} \tag{4.14}$$

Thus the error can be made arbitrarily small at the expense of more computations, but usually $P = 3$ gives sufficient accuracy for micromagnetic calculations [10] as shown in Fig. 4.3.

4.1.5 Spherical Harmonics

Spherical harmonics are the function of two variables, θ and ϕ, on the surface of a sphere and are defined in terms of associated Legendre functions [11].

$$Y_l^m(\theta,\phi) = \sqrt{\frac{(l-|m|)!}{(l+|m|)!}}\, P_l^{|m|}(\cos\theta) e^{jm\phi} \tag{4.15}$$

The associated Legendre functions, $P_l^m(x)$, in turn are defined in terms of Legendre

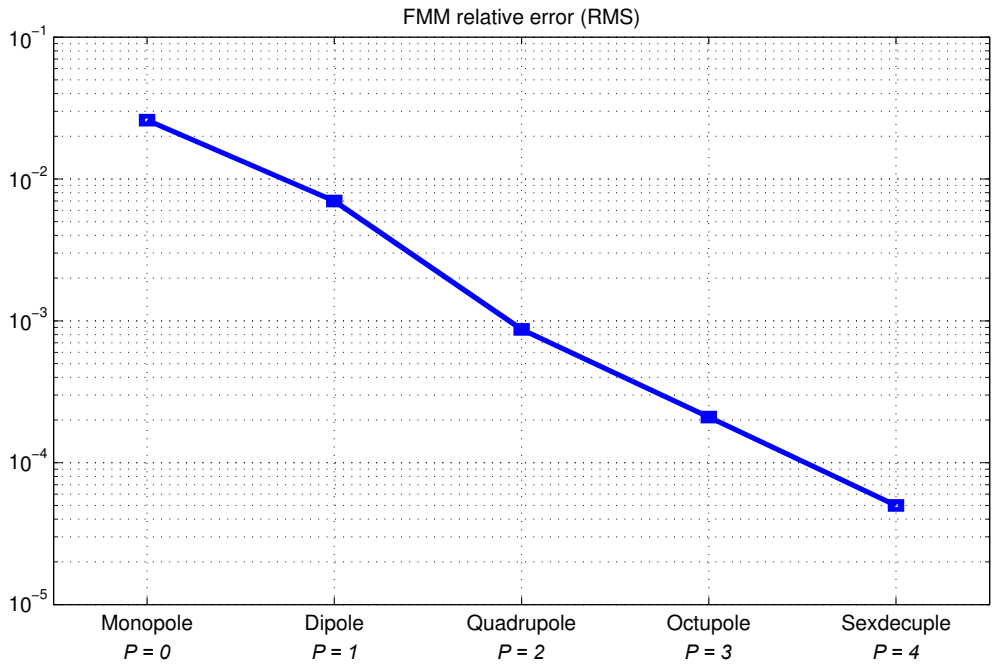


Figure 4.3: Truncation error of multipole expansion. $P = 3$ gives sufficient accuracy.

functions, $P_l(x)$, by the Rodrigues' formula [11].

$$P_l^m(x) = (-1)^m(1-x^2)^{m/2}\frac{d^m}{dx^m}P_l(x) \tag{4.16}$$

$$P_l(x) = \frac{1}{2^l l!}\frac{d^l}{dx^l}(x^2-1)^l \tag{4.17}$$

4.2 Algorithmic Description

In this section, FMM algorithm in 2-dimensions will be described using the equations of section 4.1. We are given a 2-D *computational domain* with N *mesh points*, where each point can possibly contain a charge. Our goal is to calculate potential at each mesh point in the domain.

4.2.1 Hierarchical Subdivisions

We recursively subdivide the computational domain into 4 equal parts until it reaches the refinement level of individual mesh points to get a hierarchically subdivided computational domain. In formal terms, we start with the whole domain, let's call it *refinement level* 0. Then we subdivide it into 4 equal subdomains to get to level 1. We keep subdividing the

existing subdivisions at each level until we reach at level $\log_4 N$, after which no further subdivisions are possible since we have reached the refinement level of individual mesh points. Note that at refinement level L, there are 4^L *FMM cells*, each containing $\frac{N}{4^L}$ mesh points.

4.2.2 Clustering of Charges

We index mesh points inside a FMM cell with variable i and let the charge at mesh point i be q_i. We also define $\mathbf{r}_i(r_i, \theta_i, \phi_i)$ as a vector from center of the FMM cell to the mesh point i. We form clusters of charges inside each FMM cell and represent them by multipole coefficients. The multipole coefficients (M_l^m) for a given FMM cell are calculated from the charge distribution inside the cell using Eq. (4.12).

4.2.3 Far-field Potential

Let's call the FMM cell containing the cluster of charges under consideration as the *source cell* and define $\mathbf{r}(r, \theta, \phi)$ as a vector from center of the source cell to any point in space. The multipole coefficients, representing cluster of charges in the source cell, can now be used to evaluate potential contribution from their cluster to any point $\mathbf{r}$ in the region that excludes the extent of source cell. From Fig. 4.4, it can be seen that this region includes all the FMM cells in the computational domain except those that are immediately neighboring the source cell. We define a subset of this region, let's call it the *interaction region*, as the children of the neighbors of the source cell's parent excluding source cell's immediate neighbors. See Fig. 4.4 for these topological relations. For any source cell at any refinement level in the computational domain, there can be at most 27 *interaction cells* in such an interaction region.

We limit the evaluation of potential contributions only to these interaction cells, because FMM cells exterior to this interaction region have already been taken into account during the previous coarser refinement levels. Eq. (4.10) can now be used to evaluate potential contributions to all the mesh points of these interaction cells.

4.2.4 Recursion

Potential contributions to the immediate neighbors of the source cell cannot be evaluated using Eq. (4.10) because part of these neighboring cells lie within the extent of source cell,

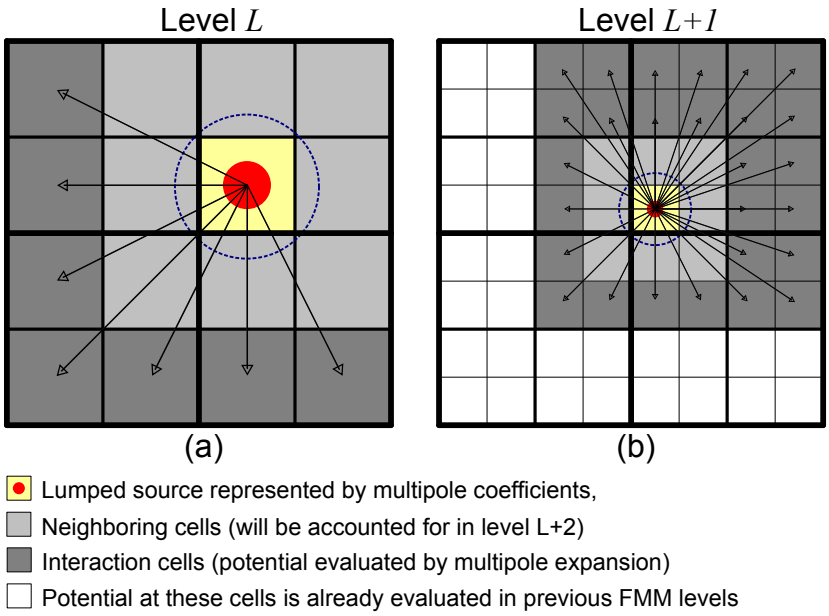


Figure 4.4: Subdivision of the 2D computational domain in the FMM algorithm. (a) Potential evaluation at level L from one cell. (b) Potential evaluation at level $L+1$ from one of the child cells of active cells in the previous step.

where multipole expansion is not valid. Contributions to these cells will be evaluated in the next finer level of refinement and this is where recursion comes into the FMM algorithm.

4.2.5 Near-field Potential

At the last refinement level ($\log_4 N$), there are no finer refinement levels to take care for the source cell's neighbors. Potential contributions to them now have to be evaluated directly according to Eq. (4.3), which is the only stage of the FMM algorithm that uses direct pairwise evaluations. But since each source cell can have a maximum of 8 neighbors, only $8N$ such evaluations are required at the finest level and thus scale with $O(N)$.

4.3 Pseudocode

4.3.1 Recursive

Recursive pseucode, which is closer to the given description of FMM, is given below.

```
function FMM(source_cell)
  CALCULATE_MULTIPOLE_COEFFICIENTS(source_cell)
  for interaction_cell = 1:27
    EVALUATE_POTENTIAL_BY_MULTIPOLE SERIES(interaction_cell)
  end
  if source_cell.level == log_4(N)
    for neighboring_cell = 1:8
      EVALUATE_POTENTIAL_DIRECTLY(neighboring_cell)
    end
  else
    SUBDIVIDE_IN_4(source_cell)
    for c = 1:4
      FMM(source_cell.child[c])  /* Recursion */
    end
  end
end
```

4.3.2 Iterative

Iterative pseucode, which helps in complexity analysis, is given below.

```
for level = 0:log_4(N)
  for source_cell = 1:4^level
      CALCULATE_MULTIPOLE_COEFFICIENTS(source_cell)
      for interaction_cell = 1:27
          EVALUATE_POTENTIAL_BY_MULTIPOLE SERIES(interaction_cell)
      end
  end
end
for level = log_4(N)
  for source_cell = 1:4^level
      for neighboring_cell = 1:8
          EVALUATE_POTENTIAL_DIRECTLY(neighboring_cell)
      end
  end
end
```

4.4 Complexity Analysis

Refer to the iterative pseucode from section 4.3.2. We determine the complexity, first for each source cell, then for each level and finally for the whole computational domain. Consider a source cell at refinement level L containing $\frac{N}{4^L}$ mesh points. For multipole series truncation at $l = P$, we need $(P+1)^2$ multipole coefficients. So the number of operations required to calculate all multipole coefficients is given by

$$(P+1)^2 \frac{N}{4^L} \tag{4.18}$$

Similarly the number of operations required to evaluate multipole expansions for $\frac{N}{4^L}$ mesh points in 27 interaction cells is given by

$$27(P+1)^2 \frac{N}{4^L} \tag{4.19}$$

There are 4^L cells at refinement level L, so the number of operations used during level

L is given by

$$\sum_{s=1}^{4^L} \left((P+1)^2 \frac{N}{4^L} + 27(P+1)^2 \frac{N}{4^L} \right) \tag{4.20}$$

$$= 4^L \left(28(P+1)^2 \frac{N}{4^L} \right) \tag{4.21}$$

$$= 28(P+1)^2 N \tag{4.22}$$

Finally there are $(\log_4 N+1)$ refinement levels in the FMM algorithm, hence the number of operations used during all the levels is given by

$$\sum_{L=0}^{\log_4 N} 28(P+1)^2 N \tag{4.23}$$

$$= 28(P+1)^2 N \log_4 N \tag{4.24}$$

An additional $8N$ operations are required to calculate direct pairwise potential cotributions at the finest level. Thus the total instruction count for FMM algorithm is given by

$$28(P+1)^2 N \log_4 N + 8N \tag{4.25}$$

Hence the asymptotic running time (complexity) of FMM algorithm is

$$O\left(28(P+1)^2 N \log_4 N + 8N\right) \tag{4.26}$$

$$= O\left(N \log N\right) \tag{4.27}$$

The reason for this reduced complexity as compared to $O(N^2)$ for direct summation is that the interactions between far-lying mesh points are evaluated using multipole expansions on large subdomains. Interactions between closer-lying regions are also accounted for with multipole expansions at finer levels of hierarchy and using smaller subdomains for the multipole expansion. And only potential contributions from immediate neighbors at the finest level are calculated with direct pairwise calculation.

4.5 Implemention

The FMM algorithm requires a lot of book-keeping and is much harder to implement than direct pairwise summation or Fast Fourier Transform (FFT).

4.5.1 Data Structures

Since we use FMM to calculate potential, it is invoked at least once (upto four times for 4th order Runge-Kutta, see chapter 5) during each time iteration of a dynamic simulation. To save running time during thousands of FMM invocations for neighbor finding, interaction list building, and other topological operations, we save all the hierarchical subdivisions along with their topological relations in a tree structure in memory. Since none of the topological relations can change over time, this scheme is expected to save running time by reducing computations at the expense of using more memory.

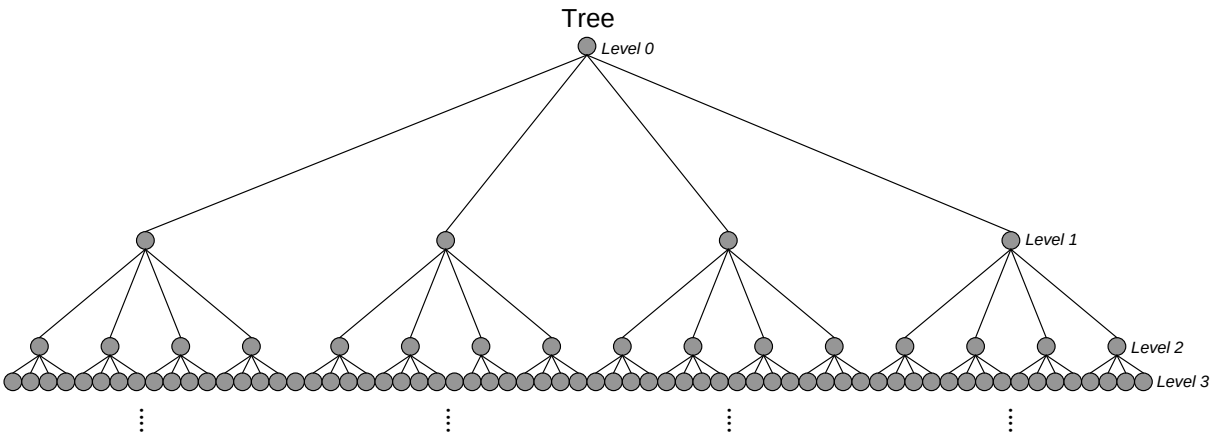


Figure 4.5: FMM tree structure.

Fig. 4.5 shows first few levels of the tree structure. Each node represents a subdivision (FMM cell) in the computational domain. The children of a node represent further subdivisions at the next finer level. From this tree structure, there is no obvious way to find neighbors and to build the interaction list. It is accomplished by using hierarchical indexing scheme [8] to number each FMM cell as shown in Fig. 4.6. This indexing scheme can keep track of spatial coordinates (x, y) of each cell as shown in Fig. 4.7 and hence can extract topological relations among subdivisions.

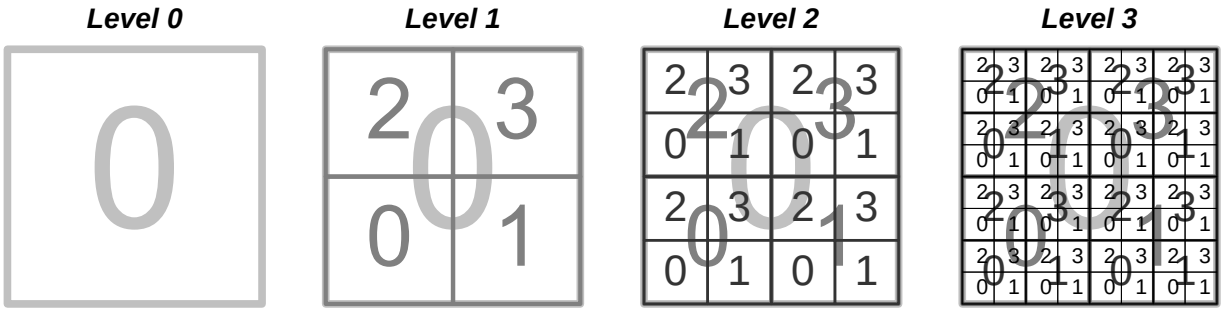


Figure 4.6: FMM hierarchical indexing based on tree structure.

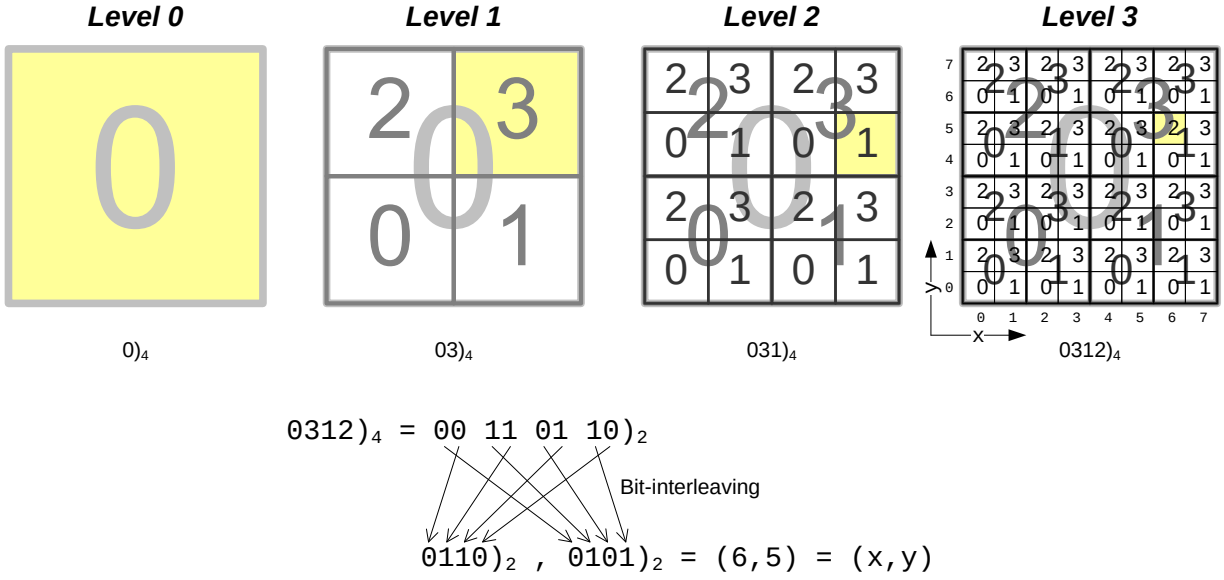


Figure 4.7: Extraction of spatial coordinates with hierarchical indexing.

4.5.2 Custom Data Types

Since the FMM algorithm makes heavy use of 3D vectors and complex numbers, we implemented custom data types (C++ classes) to effeciently represent these quantities and to write readable code. Examples of comlex quantities are multipole coefficients and spherical harmonics, while that of 3D vectors are the space vectors to represent mesh points in the computational domain.

4.5.3 Mathematical Functions

The associated Legendre functions, $P_l^{|m|}(x)$, are implemented as a lookup table in l and m but computed in x. This is because x is a continuous variable in the interval $[-1, 1]$ while

l and m take on only discrete values from a small finite set $\{l \times m\}$, where

$$l = \{0, 1, 2, \ldots, P\} \tag{4.28}$$

$$m = \{0, 1, 2, \ldots, l\} \tag{4.29}$$

For the usual case of $P = 3$,

$$|\{l \times m\}| = \sum_{l=0}^{P} \sum_{m=0}^{l} (1) \tag{4.30}$$

$$= \frac{(P+1)(P+2)}{2} \tag{4.31}$$

$$= 10 \tag{4.32}$$

The factorial function is also implemented as a lookup table because the maximum number for which we ever need a factorial is $[\max(l + |m|)] = 2P$, which for $P = 3$ is just 6.

4.5.4 Extension of 2D Algorithm to 3D

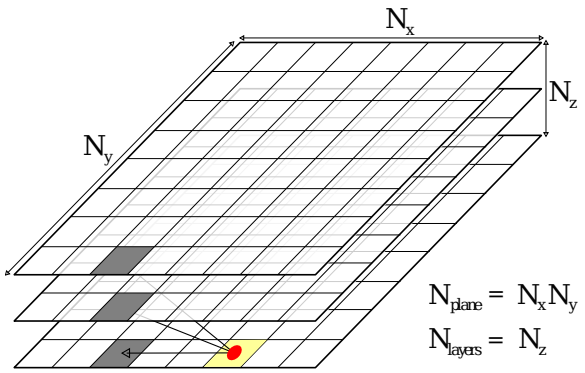


Figure 4.8: 2D FMM for 3D structures.

As nanomagnet logic devices (the first target of this simulator) are planar structures, we implemented the FMM algorithm in 2D and extended it to 3D in a brute force manner. The structure is divided as a set of planar layers, and the hierarchical subdivisions are applied separately to each layer in turn. The potential contributions from the source cell of the current layer to the interaction cells of all the layers are evaluated and thus summed up to the 3D volume as shown in Fig. 4.8.

This method scales with $O(N_{\text{layer}}^2 N_{\text{plane}} \log N_{\text{plane}})$, where N_{layer} is the number of layers and N_{plane} is the number of mesh elements per layer. For typical structures of concern, nanomagnetic logic structures for example, $N_{\text{layer}} \approx 10$ and $N_{\text{plane}} \approx 10^6$. Thus this seems to be an optimal solution.

4.6 Accuracy

As shown in Fig. 4.9, the accuracy of FMM algorithm is of the order of 10^{-4}. Even the maximum error is 10^{-3} for $P = 3$, which serves our purpose well. As can be noticed from Fig. 4.9(d), the error is maximum or minimum at the corners of the hierarchical FMM cells. This is due the fact that multipole expansion is more accurate at larger distances from the charge cluster than at nearer points.

4.7 Performance

The FMM algorithm carries a significant overhead. It is time-consuming to calculate the $(P + 1)^2$ multipole coefficients for each subdivision and then to use series expansion to evaluate potential contributions. It requires careful indexing and book-keeping to consider all interactions exactly once and exactly at the appropriate level of hierarchy. Thus the FMM is justified only for relatively large problems where its $O(N \log N)$ running time pays off while that of direct calculation ($O(N^2)$) becomes prohibitively huge (see Fig. 4.10).

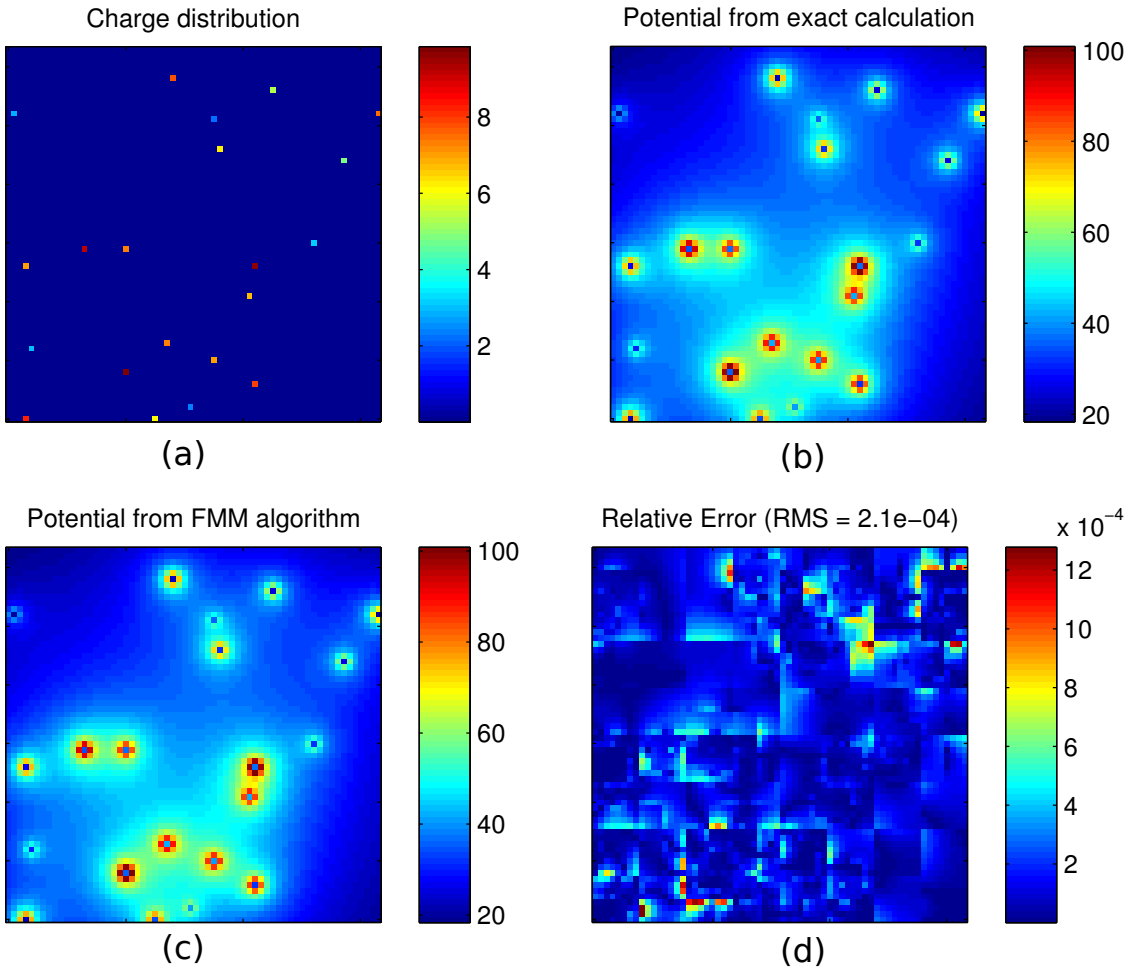


Figure 4.9: Accuracy of FMM compared with direct summation of potential. (a) Random charge distribution. (b) Potential from direct summation. (c) Potential from FMM. (d) Relative error between the two potentials.

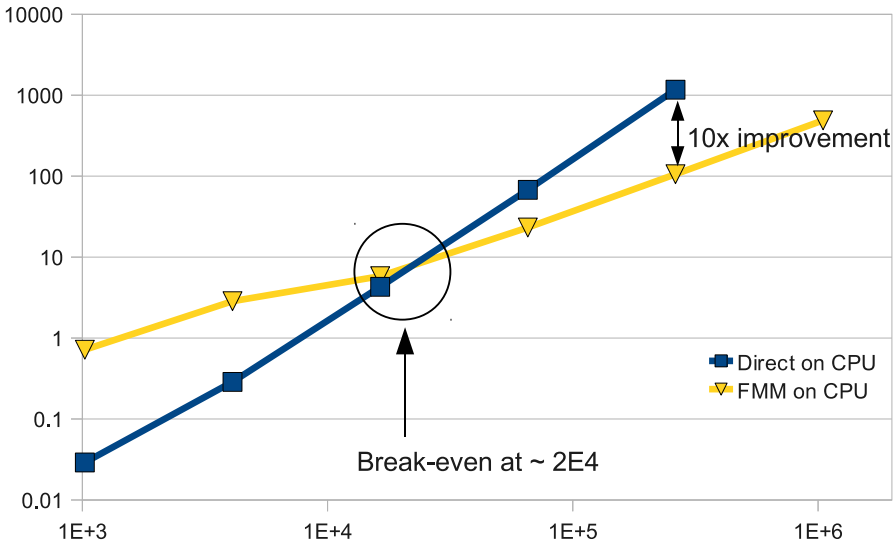


Figure 4.10: Running time for FMM versus direct summation. FMM runs faster for problems larger than $2 \cdot 10^4$ mesh points due to its $O(N \log N)$ complexity.

Chapter 5

Landau-Lifshitz-Gilbert Equation Solver

As mentioned already in section 2.2, LLG equation governs the time-evolution of magnetization vectors in a magnetic material. The equation is repeated here for convenience.

$$\frac{\partial}{\partial t}\mathbf{M}(\mathbf{r},t) = -\gamma\mathbf{M}(\mathbf{r},t) \times \mathbf{H}_{\text{eff}}(\mathbf{r},t) - \frac{\alpha\gamma}{M_s}\mathbf{M}(\mathbf{r},t) \times \Big(\mathbf{M}(\mathbf{r},t) \times \mathbf{H}_{\text{eff}}(\mathbf{r},t)\Big) \tag{5.1}$$

where all the quantities in LLG equation have already been described in section 2.1 and 2.2.

LLG equation is a system of non-linear partial differential equations. But since all the space derivates are embedded in the the Laplacian, divergence and gradient operations which are used only during the calculation of effective field ($\mathbf{H}_{\text{eff}}$) separately, we treat the LLG equation as a system of ordinary differential equations (ODE's) in time and solve it as any other initial value problem (IVP).

5.1 Euler's Solver

For an IVP, the value of dependent variable ($\mathbf{M}$) at the next time instant can be calculated from its value and derivative at current time using the straightforward point-slope formula.

$$\mathbf{M}(\mathbf{r},t+\Delta t) = \mathbf{M}(\mathbf{r},t) + \frac{\partial}{\partial t}\mathbf{M}(\mathbf{r},t)\cdot\Delta t \tag{5.2}$$

Eq. (5.2) is called the Euler's method for solving IVP and is illustrated in Fig. 5.1. It

is applied at each time-step to get the value for the next time-instant, and that's why it is also sometimes termed as *time-marching* method.

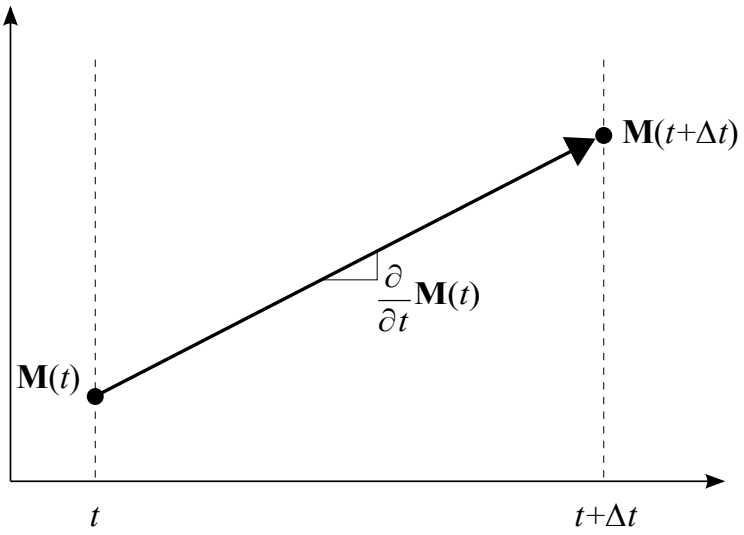


Figure 5.1: Euler's method for solving an IVP.

5.2 4th Order Runge-Kutta Solver

5.2.1 Overview

We chose to use the 4th order Runge-Kutta method for solving LLG equation due to its much better numerical stability. The basic idea is that instead of using just one slope to estimate the value at the next time instant, we use four slopes and use their weighted average for the purpose. That's why the method is called 4th order and although it requires 4 updates to the effective field for a single time-step, still it behaves better and more stable than the Euler's method with a 4 times smaller time-step.

5.2.2 Mathematical Description

Let us denote the right hand side of Eq. (5.1) by

$$\frac{\partial}{\partial t}\mathbf{M}(\mathbf{r},t) = f(\mathbf{M}(\mathbf{r},t),t) \tag{5.3}$$

Then according to the Runge-Kutta method, the value of $\mathbf{M}$ at the next time instant

can be calculated from its value at the current time and an effective slope ($\mathbf{k}$).

$$\mathbf{M}(\mathbf{r}, t + \Delta t) = \mathbf{M}(\mathbf{r}, t) + \mathbf{k} \cdot \Delta t \tag{5.4}$$

$$\mathbf{k} = \frac{\mathbf{k}_1}{6} + \frac{\mathbf{k}_2}{3} + \frac{\mathbf{k}_3}{3} + \frac{\mathbf{k}_4}{6} \tag{5.5}$$

where $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ and $\mathbf{k}_4$ are the slopes at different points along the interval $[t, t + \Delta t]$ as shown in Fig. 5.2 and defined below.

$$\mathbf{k}_1 = f\Big(\mathbf{M}(\mathbf{r}, t), t\Big) \tag{5.6}$$

$$\mathbf{k}_2 = f\Big(\mathbf{M}(\mathbf{r}, t) + \mathbf{k}_1 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\Big) \tag{5.7}$$

$$\mathbf{k}_3 = f\Big(\mathbf{M}(\mathbf{r}, t) + \mathbf{k}_2 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\Big) \tag{5.8}$$

$$\mathbf{k}_4 = f\Big(\mathbf{M}(\mathbf{r}, t) + \mathbf{k}_3 \Delta t, t + \Delta t\Big) \tag{5.9}$$

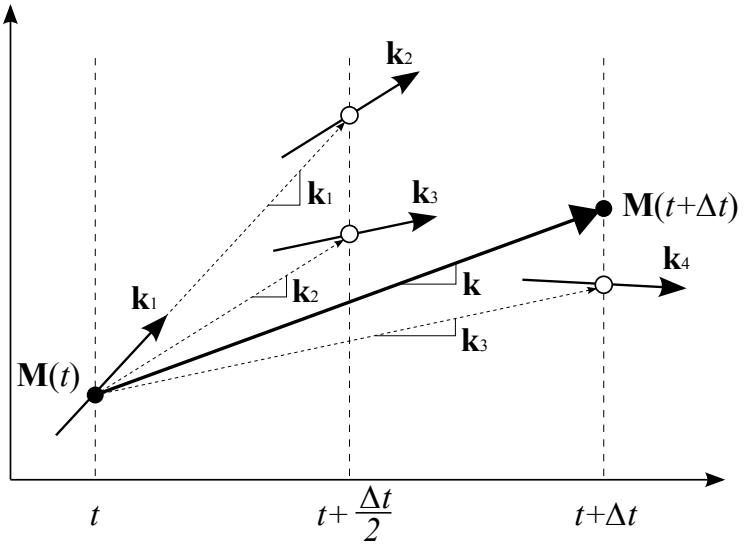


Figure 5.2: 4th order Runge-Kutta method. The slope $\mathbf{k}$ is the weighted average of $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ and $\mathbf{k}_4$.

The above steps of Runge-Kutta method are applied at each time-step to evolve the magnetization, $\mathbf{M}(\mathbf{r}, t)$, over time.

5.3 Adaptive Time-stepping

The choice of the time-step (Δt) in IVP solvers has significant impact on their accuracy, stability and running time. The simplest approach is to always use a fixed and conservative time-step. Though very stable, this approach wastes so much of the precious running time of the simulation by not making large strides during the periods of low dynamic activity. Usually, one is only intersted in capturing fine details during the periods of high dynamic activity, while low dynamic activity periods are desired to be captured with less details to save running time as well as storage requirements.

5.3.1 Overview of Algorithm

We use a simple method to decide an optimal time-step on the fly. We monitor the maximum torque on magnetization vectors in the material and use this information to scale up or down the time-step accordingly. If the maximum torque is growing, which indicates that the system is approaching a period of high dynamic activity, we reduce the time-step in order to capture finer details of time-evolution of magnetization. On the other hand, if the maximum torque is shrinking, which indicates that the system is approaching a period of low dynamic activity, we increase the time-step in order to make a few large strides to quickly pass through the periods of less interesting time-evolution of magnetization.

5.3.2 Mathematical Description

At each time-step, we sample a quantity from the system that is the represntative of torque on magnetization vectors in the material. Let this quanity be called τ.

$$\tau(\mathbf{r}, t) \quad = \quad \frac{\mathbf{M}(\mathbf{r}, t) \times \mathbf{H}_{\text{eff}}(\mathbf{r}, t)}{M_s^2} \tag{5.10}$$

where M_s is the saturation magnetization of the material. We then take the maximum out of all the torques in the material,

$$\tau_{\max}(t) \quad = \quad \max_{\mathbf{r}} \quad \tau(\mathbf{r}, t), \tag{5.11}$$

and use this quantity to scale the time-step.

$$\Delta t(t) \;=\; \frac{k}{\tau_{\max}(t)} \tag{5.12}$$

where k is a parameter in the range of 10^{-14} to 10^{-12}. The inverse relation between Δt and $\tau_{\max}(t)$ is also depicted in Fig. 5.3.

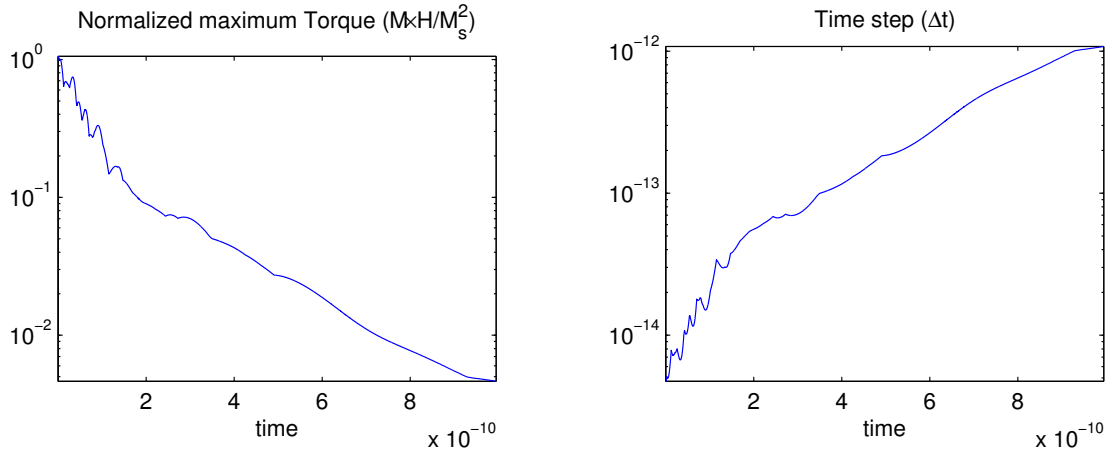


Figure 5.3: Time-evolution of torque and time-step.

When using Eq. (5.12) to choose the time-step, sometimes it so happens that spurious oscillations are introduced into the system when it is close to settling down. To prevent these artifacts we also monitor the magnetostatic and exchange energies at each time step.

$$E_{\text{demag}}(t) \;=\; -\int_{\text{volume}} \mathbf{M}(\mathbf{r},t)\cdot\mathbf{H}_{\text{demag}}(\mathbf{r},t)\; d\mathbf{r} \tag{5.13}$$

$$E_{\text{exch}}(t) \;=\; -\int_{\text{volume}} \mathbf{M}(\mathbf{r},t)\cdot\mathbf{H}_{\text{exch}}(\mathbf{r},t)\; d\mathbf{r} \tag{5.14}$$

Since the sum of these energies must always descrease over time (Fig. 5.4), we reject any time-step that would do otherwise. When this happens, we then recompute it with a forcibly reduced conservative time-step. This is similar to the criterion used in the well established OOMMF code [12].

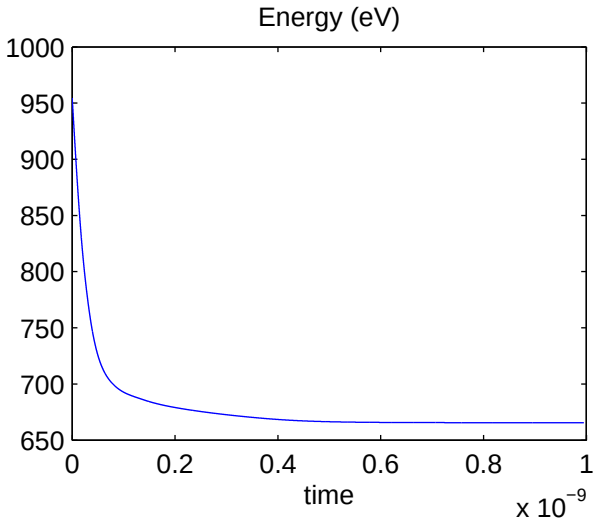


Figure 5.4: Time-evolution of energy ($E_{\mathrm{demag}} + E_{\mathrm{exch}}$).

Chapter 6

Simulation Examples

In this chapter, we discuss two simulation expamples and compare the results with literature and well established codes to demonstrate that our code is working correctly.

6.1 Ordering of Nanowires

The simplest test structure to demonstrate is the antiferromagnetic ordering in nanomagnet-wires. Fig. 6.1 shows two wires of different lengths that we investigated. We exposed them to an external ramping field in order to aid switching in the correct order and let them settle for a few nanoseconds.

Our simulation results showed that the shorter wire got ordered perfectly while the longer one had 2 imperfections out of 6 field coupling points. These results were completely in agreement with OOMMF simulations and earlier experimental results.

6.2 Phase Transition of Nanomagnets

The multi-domain to single-domain transition is one of the most characteristic phenomena of nanomagnetic behavior. For small magnets; exchange fields, which are reponsible to align the magnetizations with each other, are stronger and drive the magnet into a homogeneously magnetized (single-domain) state. For larger magnets, the demagnetization field takes over the exchange field and promotes the formation of a vortex state. The transition between the two states takes place at a well defined magnet size. A value closely in agreement established codes and experimental data can prove the accurate calculations of

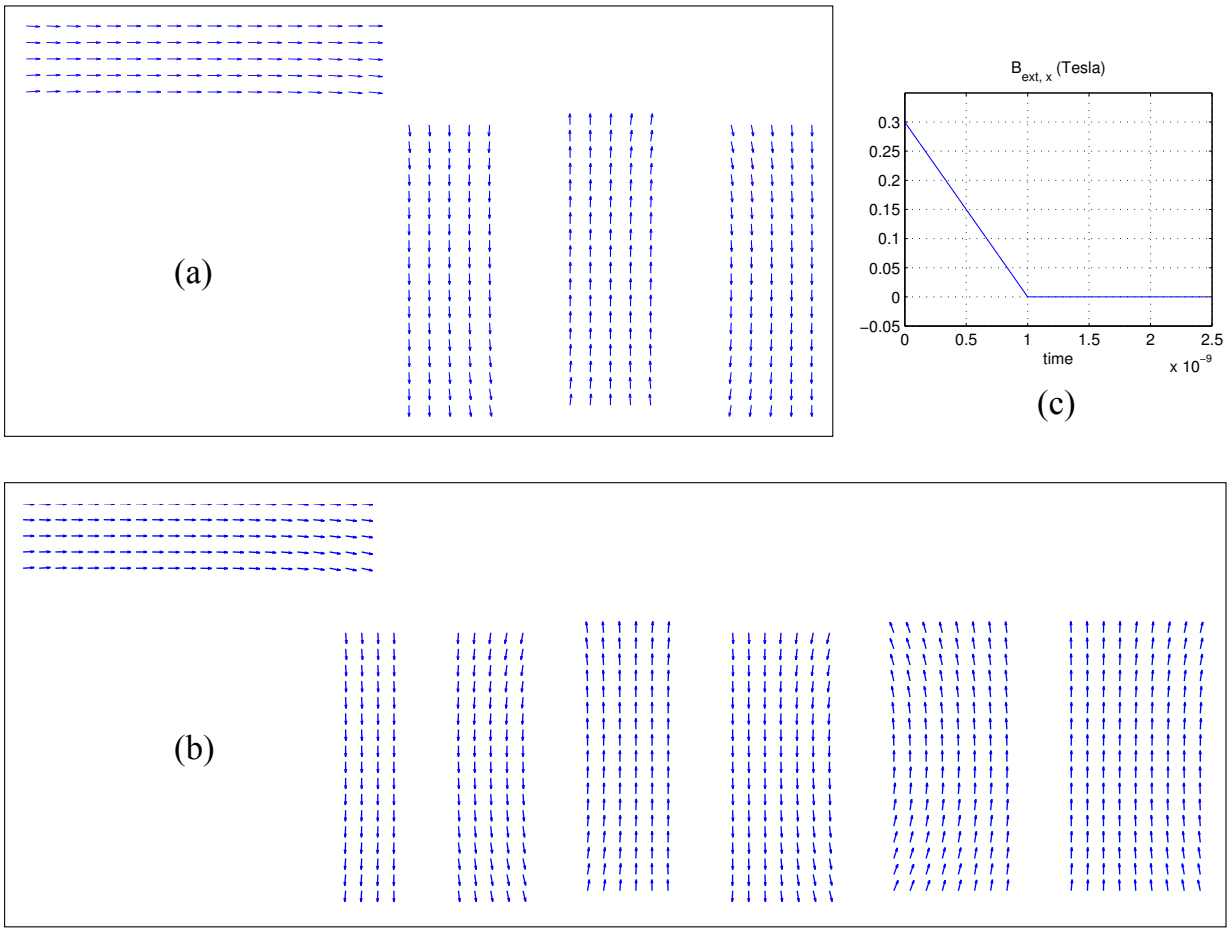


Figure 6.1: Ordering of nanowires. (a) Perfectly ordered short wire. (b) Imperfectly ordered long wire. (c) Applied external field to aid in the switching of magnets.

magnetostatic and exchange fields as well as the LLG solver. So this is a very comprehensive test of any micromagnetic simulator as it tests almost all the physics in there.

We investigated a 5 nm thick, square-shaped permalloy nanomagnet of different sizes. For each magnet size, we performed two simuations – (1) starting from a single domain magnetization state and calculated the magnetic energy after letting it settle, and (2) starting from a vortex magnetization state and calculated the magnetic energy after letting it settle.

It is well establshed that when starting from a random state or in presence of thermal noise, a magnet will always converge to the minimum energy state. Thus the phase

transition occurs when the energy difference between the two relaxed states is zero.

$$\begin{aligned} E_{\text{1domain}} - E_{\text{vortex}} = \Delta E &< 0 \ ; \text{ Single domain state} \\ \Delta E &= 0 \ ; \text{ Phase transition} \\ \Delta E &> 0 \ ; \text{ Vortex state} \end{aligned}$$

This energy difference is shown in Fig. 6.2, as a function of the magnet size. The figure also shows that the phase transition occurs at 185 nm, in good agreement with the results given by well-established codes, such as OOMMF [12]. This benhcmark proves that all the components of our simulator code are working correctly.

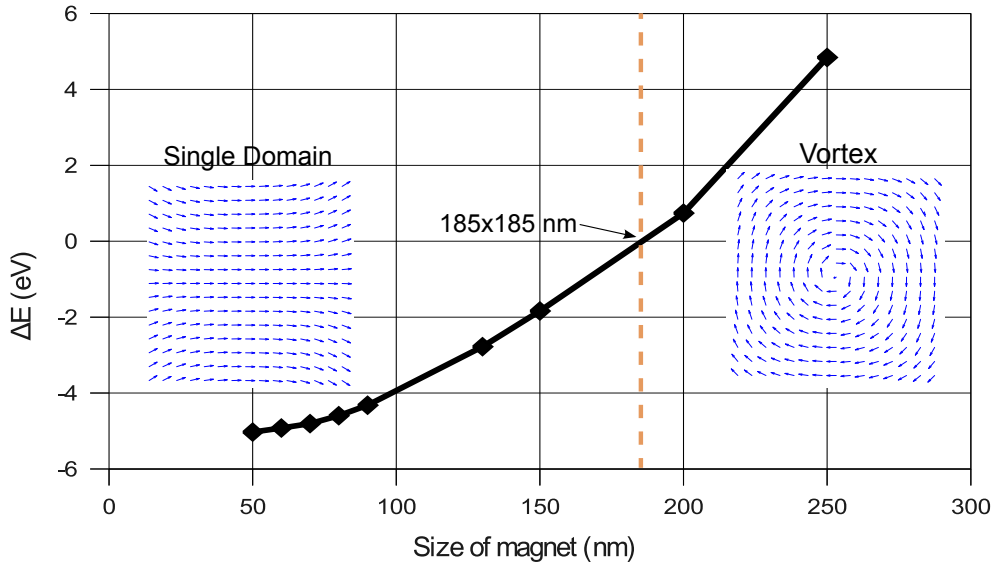


Figure 6.2: Phase transition between single-domain and vortex state of a square-shaped permalloy nanomagnet. Energy difference ($\Delta E = E_{\text{1domain}} - E_{\text{vortex}}$) between relaxed single-domain and vortex states is plotted as a fucntion of magnet size and shows that the transition occurs at around 185×185 nm.

Chapter 7

Parallelization on GPU

In this chapter, we discuss about the parallelization of computationally intensive parts of the simulator on the GPU. We also show performance benchmarks for the parallelized code. To put the benchmark figures in perspective, we used a workstation having an Intel Xeon X5650 CPU (6 cores) with 24 GB of main memory and an NVIDIA Tesla C2050 GPU (448 cores) with 3GB of its own DRAM. Although the CPU is 6-core but for the purpose of benchmarking, we only compared the performance of parallelized GPU code against that of single-threaded CPU code.

7.1 Parallelization of the LLG Solver

The LLG solver, which is a 4th order Runge-Kutta method (c.f. Sec. 5.2), was easily parallelized by domain decomposition to achieve a speed-up of 100x (c.f. Fig. 7.1). In addition, all the other local operations like exchange field (Eq. (2.4)), gradient and divergene operations can be likewise parallelized. However, for any non-trivial sized problem (number of mesh elements > 100), the potential calculation (Eq. (3.7)) is by far the most time consuming part (more than 90% of the running time) and so we focused on the GPU implementation of this part only. We used OpenMP on the CPU to parallelize all the local operations including the LLG solver.

7.2 Parallelization of the Direct Summation Method

To implement Eq. (3.7), we need two nested loops – sources ($\mathbf{r}_i$) and observers ($\mathbf{r}$), and their order is also interchangeable. NVIDIA GPU architecture also offers two levels of parallelism

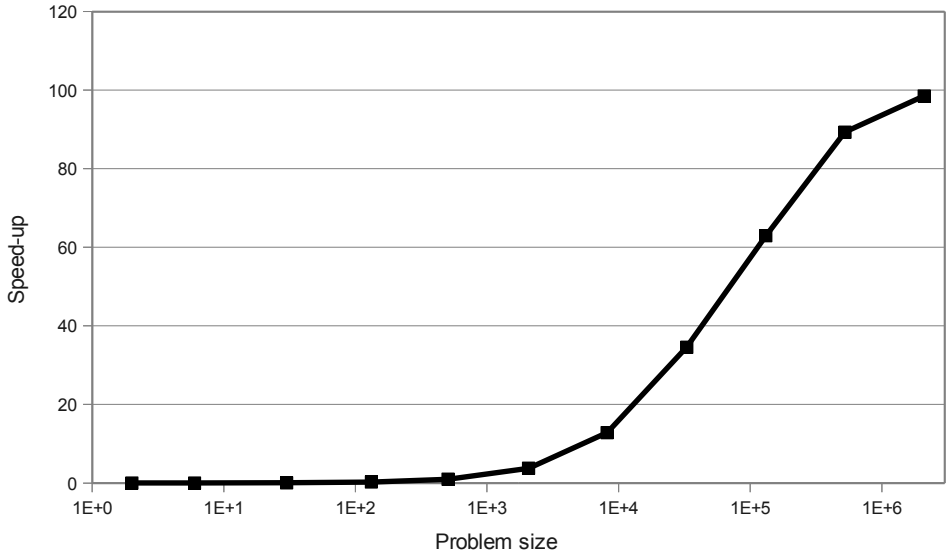


Figure 7.1: Speed-up of LLG solver on GPU.

– multiprocessor (block) level and thread level, so this problem maps straightforwardly to the GPU.

At the observer ($\mathbf{r}$) level, we have to sum potential contributions from N sources,

$$\Phi(\mathbf{r}) = \sum_{\mathbf{r}_i \in \text{volume}} \frac{q(\mathbf{r}_i)}{|\mathbf{r} - \mathbf{r}_i|} \tag{7.1}$$

and to execute for each source ($\mathbf{r}_i$), a *read-modify-write* instruction of the form,

$$\Phi(\mathbf{r}) = \Phi(\mathbf{r}) + \frac{q(\mathbf{r}_i)}{|\mathbf{r} - \mathbf{r}_i|} \;\; ; \;\; \mathbf{r}_i \in \text{volume} \tag{7.2}$$

This read-modify-write instruction requires data sharing and synchronization among parallel workers. We map sources ($\mathbf{r}_i$) to threads and observers ($\mathbf{r}$) to blocks because there is no efficient way to share data among the blocks, while doing so among the threads is highly efficient. For summing up potential contributions from N sources (threads) to a single observer, parallel reduction algorithm [13] is utilized, which exploits the architectural features of the GPU efficiently (Fig. 7.2).

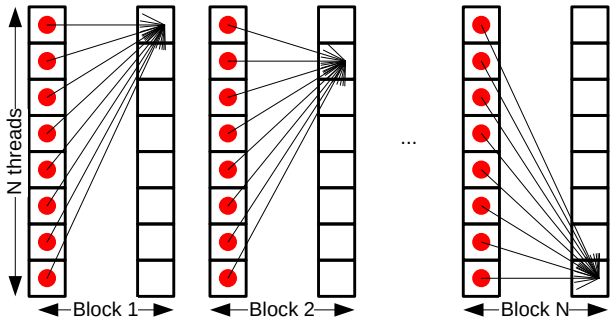


Figure 7.2: Parallel reduction algorithm.

7.3 Paralellization of the FMM

Complexity analysis from Sec. 4.4 shows that running time of potential evaluation through multipole expansion (Eq. (4.10)) is at least $27N_{\text{layers}}$ times greater than that of coefficient calculation (Eq. (4.12)). So as a first step we paralellized only this part of the FMM algorithm on GPU. For work sharing scheme, it is natural to distribute work over 27 interaction cells to GPU blocks, since they do not have to share data among each other. Threads within a GPU block are then mapped to individual mesh points inside the corresponding interaction cell for the evaluation of multipole expansion to contribute to their potential value.

The downside of this naive approach to FMM parallelization is that the GPU computation is launched in one of the inner nested loops of FMM algorithm, i.e. it has to be started again and again for each cell in the FMM hierarchy. Therefore several times during a single FMM run, data has to be transferred between CPU and GPU memories, which becomes a severe bottleneck. Work is in progress to eliminate this bottleneck by implementing the entire FMM algorithm on GPU.

7.4 Performance Benchmarks of the Parallelized Code

Firt of all, we verified that the parallel (GPU) and single-threaded (CPU) codes are giving the same results within the floating-point accuracy.

Direct potential calculation is straightforward to paralellize as discussed already in Sec. 7.2. Speed-up factors (as compared to the single-threaded CPU version) are shown in Fig. 7.3. As other components of the simulator are running on the CPU, the GPU computation has to be launched in each iteration step of the time loop. However, speed-up

of up to 30x can be obtained when, for large problems, the GPU launch time becomes negligible compared to the computation time.

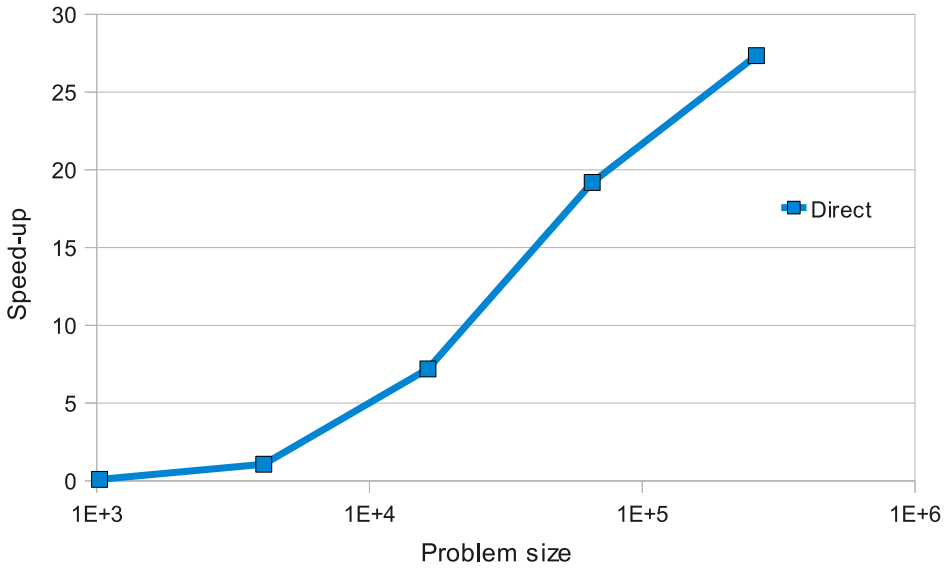


Figure 7.3: Speed-up of direct summation method on GPU. Speed-up factors of about 30x are obtained for large problems.

Fig. 7.4 shows a modest speedup for the FMM-based field solver (3x – 4x). This is due to the fact that, in the current implementation, only a fraction of the FMM algorithm is running on the GPU and that the GPU computation is launched several times even for a single potential calculation.

For part of the potential calculation, at coarser levels of FMM hierarchy, a very large number of potential values have to be evaluated using the same coefficients of the multipole expansion and the GPU is launched fewer times. These parts of the FMM algorithm show a 10x speed-up on GPU. This suggests that implementing the entire FMM algorithm on GPU would accelerate the calculation at least that much in the future.

Yet in this unoptimized version, the parallelized FMM has its strength for very large-sized problems ($N > 2 \cdot 10^5$), where it steeply outperforms the direct summation method (Fig. 7.5) due to its better $O(N \log N)$ complexity. For smaller problems, however, the well-paralellized direct summation method runs faster.

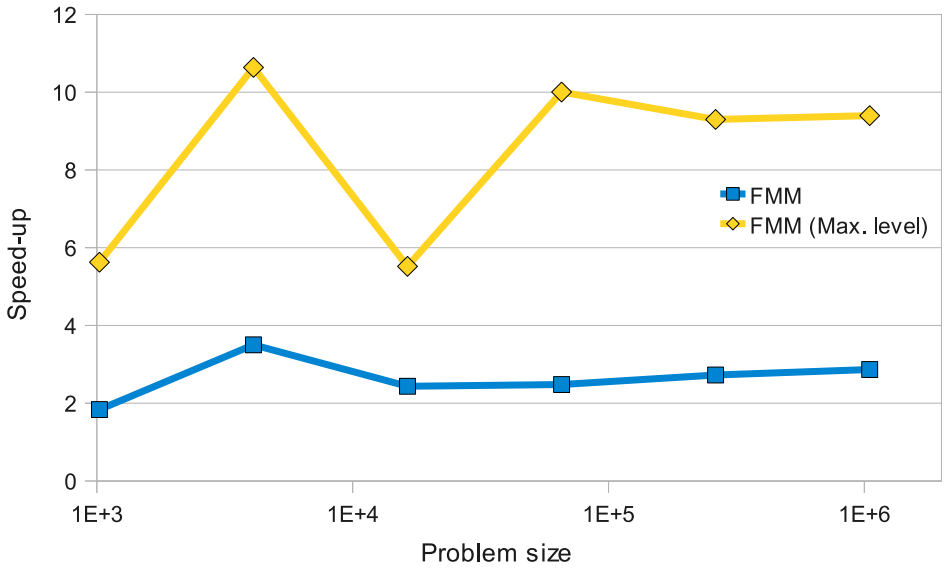


Figure 7.4: Speed-up of FMM on GPU. Speed-up factors of about 3x – 4x are achieved but are as high as 10x for the coarsest part of the FMM algorithm.

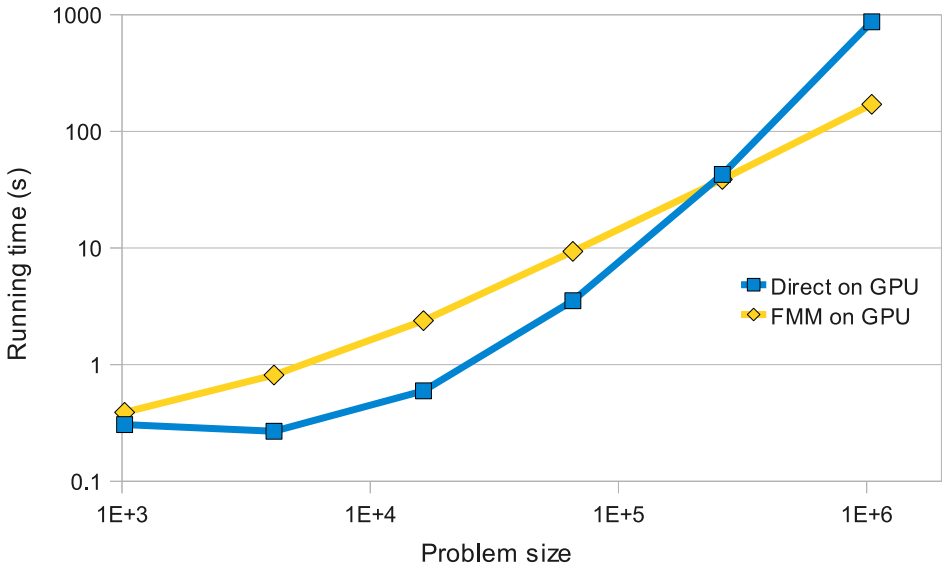


Figure 7.5: Comparison of running times of direct summation method and FMM on GPU. FMM runs faster for problems larger than $2 \cdot 10^5$ mesh points due to its $O(N \log N)$ complexity.

Chapter 8

Conclusion and Future Work

The goal of this work was to build a three-dimensional, parallelized and GPU-accelerated micromagnetic simulator which is geared towards the simulation of nanomagnetic computing devices. We started the work 'from scratch and implemented all the algorithms without reusing program modules from any other source. The most involved part of the work is the magnetostatic potential (field) calculation. We successfully implemented two algorithms for field calculation.

1. Direct summation
2. Fast Multipole Method (FMM)

The direct potential calculation is straightforward to parallelize by domain decomposition and without much optimization we already achieved a speedup of 30x compared to a single-threaded CPU version. The FMM, however, is far more complex and so far only a fraction of the algorithm is ported on GPU this resulted in a moderate speedup of 4x.

We also tested our simulator against standard micromagnetic problems and found it to be working very accurately in agrrement with well established codes such as OOMMF [12]. Our work resulted in

1. a fully customizable, high performance micromagnetic simulator that will be the basis for future simulation work of research groups working on nanomagnetic logic devices, and
2. demonstrated the possibility of achieving significant speed-up by performing the most computationally intensive parts on calculations the GPU.

In the next phase of the work, we focus on porting the entire field-calculation routine on the GPU and optimizing the FMM algorithm in order to avoid the bottleneck of too many time-consuming memory transfers between the CPU and the GPU. We hope that speed-ups of upto 50x will be achievable for the FMM based calculation.

Bibliography

[1] Jacques E. Miltat and Michael J. Donahue. Numerical micromagnetics: Finite difference methods. In *Handbook of Magnetism and Advanced Magnetic Materials*, volume 2: Micromagnetism. John Wiley & Sons, 2007.

[2] A. Imre, G. Csaba, L. Ji, A. Orlov, G.H. Bernstein, and W. Porod. Majority logic gate for magnetic quantum-dot cellular automata. *Science*, 311 no. 5758:205 –208, 2006.

[3] György Csaba. *Computing with Field-Coupled Nanomagnets*. PhD thesis, University of Notre Dame, 2003.

[4] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879 – 899, May. 2008.

[5] Shaojing Li, B. Livshitz, and V. Lomakin. Graphics processing unit accelerated $O(N)$ micromagnetic solver. *Magnetics, IEEE Transactions on*, 46(6):2373 –2375, jun. 2010.

[6] Richard L. Coren. *Basic Engineering Electromagnetics – An Applied Approach*. Prentice-Hall, 1989. Chapter 7.

[7] Leslie Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. MIT Press, Cambridge, MA, 1998.

[8] N.A. Gumerov, R. Duraiswami, and E.A. Borovikov. Data structures, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions. Computer Science Technical Report 4458, University of Maryland, College Park, 2003.

[9] The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News*, 33(4).

[10] Gregory Brown, M. A. Novotny, and Per Arne Rikvold. Langevin simulation of thermally activated magnetization reversal in nanoscale pillars. *Phys. Rev. B*, 64(13):134422, Sep 2001.

[11] John David Jackson. *Classical Electrodynamics*. John Wiley & Sons, 3rd edition, 1999. Chapter 2 & 3.

[12] The Object Oriented MicroMagnetic Framework (OOMMF). `http://math.nist.gov/oommf`. A widely used, public-domain micromagnetic software micromagnetic program.

[13] Mark Harris. Optimizing parallel reduction in CUDA. In *NVIDIA CUDA SDK – Data-Parallel Algorithms*. 2007.