UNIVERSITY OF CALIFORNIA, SAN DIEGO

Parallel Computation with Fast Algorithms for Micromagnetic Simulations on GPUs

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Electronic Circuits and Systems)

by

Sidi Fu

Committee in charge:

Professor Vitaliy Lomakin, Chair
Professor Prabhakar R. Bandaru
Professor Eric E. Fullerton
Professor Zhaowei Liu
Professor Ross C. Walker

2016

The Dissertation of Sidi Fu is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2016

DEDICATION

To my mother Kun Zhao, my father Xiaojun Fu

and

my dear Yan Jiao.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to sincerely thank the people who have helped me throughout my graduate study at UC San Diego. Among them, I especially want to appreciate my advisor Prof. Vitaliy Lomakin. He offered me the opportunity to join the CEM group as a member of this “big family”. He not only supports me through his guidance in the research but also through the suggestions in normal life. He plays the roles of an advisor, a father and a sincere friend at the same time. His profound knowledge in magnetism, electromagnetism, numerical methods and parallel computing allows me to progress to this stage in my Ph.D. study. Prof. Lomakin also supports me for attending academic conferences and summer internships, which greatly broadened my vision of the future career.

I would also like to thank Adam Torabi, Byron Lengsfield, Terry Olsen, Roger Wood, Lei Xu, Jihoon Park, Gregory Parker and Yimin Hsu for their help during my three summer internships at HGST. They always encouraged me and made me feel at home, and I also appreciate their teaching me about the hard drive industry. I will not be able to complete the work about discrete cap model for magnetic media in Chapter 5 without them. My gratitude also goes to Brian Cabral, Vladimir Bychkovsky and Jason Carreiro for mentoring me during the internship at Facebook. Their way of thinking and hands on the projects inspired me in my research work back at UC San Diego. Also, I appreciate the opportunities to work with Prof. Eric Fullerton and Prof. Boubacar Kante as a teaching assistant in their classes.

The last but not the least, my lab-mates at the CEM group has been considerably helped me throughout my study. Shaojing Li and Ruinan Chang introduced me to the lab

and patiently answered all my questions. My great thank goes to Dr. Marko Lubarda and Dr. Marco Escobar, Qian Ding and Javier Espigares for their welcoming me and helping me with patience. I also appreciate to working and being a friend with Simon Couture, Majd Kuteifan, Marco Menarini and Iana Volvach, who comes later as my group mates. I am grateful to Weilong Cui, Matthew Hu, Sicong Yan and Philippe Scheid who worked with me during their internship at CEM group as visiting scholars.

Chapter 1, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, R. Chang, S. Couture, M. Menarini, M.A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay, V. Lomakin, "*Micromagnetics on High-performance Workstation and Mobile Computational Platforms*". Journal of Applied Physics, vol. 117, no. 17, pp. 17E517, 2015.

Chapter 2, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, W. Cui, M. Hu, R. Chang, M.J. Donahue, V. Lomakin, "*Finite Difference Micromagnetic Solvers with Object Oriented Micromagnetic framework (OOMMF) on Graphics Processing Units*," IEEE Transactions on Magnetics, vol. 52, no. 4, pp. 1-9, 2016. S. Fu, R. Chang, S. Couture, M. Menarini, M.A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay, V. Lomakin, "*Micromagnetics on High-performance Workstation and Mobile Computational Platforms*". Journal of Applied Physics, vol. 117, no. 17, pp. 17E517, 2015.

Chapter 3, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of

this paper: S. Fu, W. Cui, M. Hu, R. Chang, M.J. Donahue, V. Lomakin, "*Finite Difference Micromagnetic Solvers with Object Oriented Micromagnetic framework (OOMMF) on Graphics Processing Units*," IEEE Transactions on Magnetics, vol. 52, no. 4, pp. 1-9, 2016.

Chapter 4, in part, is currently being prepared for submission for publication of the material, where the dissertation author was the primary investigator and author of this paper: S. Fu, R. Chang, I. Volvach, M. Kuteifan, S. Couture, M. Menarini, V. Lomakin, "*Block Diagonal preconditioner for implicit time integration in finite element micromagnetic solvers*".

Chapter 5, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, L. Xu, V. Lomakin, A. Torabi, B. Lengsfield. "*Modeling perpendicular magnetic multilayered oxide media with discretized magnetic layers*." IEEE Transactions on Magnetics, vol. 51, no. 11, pp. 1-4, 2015.

VITA

2011 B.S. in Micromagnetics, Peking University

2013 M.S. in Electrical Engineering (Electronic Circuits and Systems), University of California, San Diego

2016 Ph.D. in Electrical Engineering (Electronic Circuits and Systems), University of California, San Diego

PUBLICATIONS

S. Fu, W. Cui, M. Hu, R. Chang, M.J. Donahue, V. Lomakin, "*Finite Difference Micromagnetic Solvers with Object Oriented Micromagnetic framework (OOMMF) on Graphics Processing Units*," IEEE Transactions on Magnetics, vol. 52, no. 4, pp. 1-9, 2016.

S. Fu, L. Xu, V. Lomakin, A. Torabi, B. Lengsfield. "*Modeling perpendicular magnetic multilayered oxide media with discretized magnetic layers*." IEEE Transactions on Magnetics, vol. 51, no. 11, pp. 1-4, 2015.

S. Fu, R. Chang, S. Couture, M. Menarini, M. A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay, V. Lomakin, "*Micromagnetics on high-performance workstation and mobile computational platforms*." Journal of Applied Physics, vol. 117, no. 17, pp. 17E517, 2015.

M. Kuteifan, M. Lubarda, S. Fu, R. Chang, M. A. Escobar, S. Mangin, E. E. Fullerton, V. Lomakin. "*Large exchange-dominated domain wall velocities in antiferromagnetically coupled nanowires*." AIP Advances, vol. 6, no. 4, pp. 045103, 2016.

J. Park, B. Lengsfield, R. Galbraith, R. Wood, S. Fu. "*Optimization of Magnetic Read Widths in Two-Dimensional Magnetic Recording Based on Micromagnetic Simulations*." IEEE Transactions on Magnetics, vol. 51, no. 11, pp. 1-4, 2015.

ABSTRACT OF THE DISSERTATION

Parallel Computation with Fast Algorithms for Micromagnetic Simulations on GPUs

by

Sidi Fu

Doctor of Philosophy in Electrical Engineering (Electronic Circuits and Systems)

University of California, San Diego, 2016

Professor Vitaliy Lomakin, Chair

Micromagnetics is a field of study considering the magnetization behavior in magnetic materials and devices accounting for a wide set of interactions and describing the magnetization phenomena from the atomistic scale to several hundreds of microns. Micromagnetic simulations are essential in understanding the behavior of many magnetic systems. Modeling complex structures can require a significant computational time and in

some cases, the system complexity can make simulations prohibitively long or require a prohibitively large memory.

In this thesis, we present a set of methods and their implementations that resulted in high-performance numerical micromagnetic tools for modeling highly complex magnetic materials and devices. The focus of the dissertation is on solving Landau-Lifshitz-Gilbert (LLG) equation efficiently, both with numerical methods and advanced hardware acceleration.

To understand the numerical problem to be solved, the introduction Chapter 1 addresses the LLG equation and the governing interactions involved as well as numerical modeling basics on the Finite Difference Method (FDM) and the Finite Element Method (FEM). Chapter 1 also presents a versatile micromagnetic framework, referred to as FastMag, which implements some of these methods.

Chapter 2 provides a detailed description of computing based on Graphics Processing Units (GPUs). The history of GPU programming model and the programming tips serve as the basis for understanding parallel computing on GPUs. It presents applications of GPUs on various platforms to demonstrate the current mainstream usage of GPUs and their promising future development direction. Chapter 2 also summarizes applications of GPUs in micromagnetics.

Chapters 3 and 4 address two essential aspects of micromagnetic solvers: fast algorithms for computing the key interaction components and efficient time integration methods. Chapter 3 introduces a non-uniform Fourier transform (NUFFT) method, a scalar potential method, and sparse matrix-vector multiplication (SpMVM) algorithms implemented on GPUs to accelerate the magnetostatic and exchange interactions. Chapter

4 addresses basics of the time integration methods used in FastMag as well as a preconditioner to further accelerate the time integration process.

Chapter 5 presents a numerical model for the current state-of-art magnetic recording system using advanced algorithms and GPU implementations described in Chapters 2-4.

1. Introduction

1.1. Introduction to Micromagnetics

Micromagnetics predicts the sub-micrometer magnetic behaviors in various systems via numerical modeling [1]. The length-scale covered by the micromagnetics is small enough to consider sub-micrometer behaviors like magnetic domain walls but large enough to average out the atom-level behaviors [2]. Micromagnetic modeling has significant predictive power and is essential for analyzing and designing magnetic devices and systems. The modeling of magnetic recording systems (magnetic write/read heads, magnetic media in hard-drives), magnetic random-access memory (MRAM), spin-torque oscillator (STO) and other magnetic systems are important applications of micromagnetics.

Micromagnetics originates from the need to explain magnetic phenomena such as magnetic domain wall formation, domain patterns, nucleation fields, reversal modes and magnetization dynamics [3]. The magnetic moment density and relevant material parameters were introduced as continues variables by W.F. Brown in 1978 [4]. The micromagnetic modeling surged in the past 20 years driven by the strong growth of the scientific computational power.

Micromagnetic simulations can be divided into two categories: static simulations and dynamic simulations. For static simulations, the equilibrium state, the energy barrier or the energy landscape of magnetic systems are of concern. Here, the equilibrium states

refer to the stable spatial distribution of the magnetization. Methods such as energy minimization method, Nudged Elastic Band (NEB) method, and the relaxation of Landau-Lifshitz-Gilbert (LLG) equation are the important candidates to solve the static problems. As to the dynamic simulations, the dynamics of the magnetic moments in a certain system is the focus. Dynamic simulations usually are based on solving the LLG equation. In some cases, the LLG equation is modified to be able to address specific dynamic behavior types, e.g. the Landau-Lifshitz-Bloch (LLB) equation is necessary for systems in which the temperature can approach the Curie temperature [5].

To model geometries in a complex magnetic system, the system needs to be discretized into smaller sub-domains and each subdomain (or its boundaries) is assigned an unknown magnetization state. Modeling methods in Micromagnetics can be categorized into two kinds based on how the discretization is defined: Finite Difference methods (FDMs) and Finite Element methods (FEMs). FDMs discretize the geometries into uniform cubes (or bricks), while the FEMs typically use tetrahedrons. The way that FDM handles the discretization is simpler than FEM. As a result, the FDM modeling is easier to implement and the simulation often may be faster for simple systems. The accuracy of FDM is good enough for relatively simple structures like magnetic thin films. However, FEM gives more flexibility in modeling complex, e.g. non-uniform and non-regular boundaries and geometries, by utilizing tetrahedrons.

An important component in enabling the analysis and design of complex magnetic devices is the development of parallel computational codes [6]. Parallel computation is generally demonstrated on the parallel platforms such as multi-core CPUs, GPUs, and multiple computational nodes of CPUs and GPUs. Taking advantage of the fact that

certain amount of the compute-intensive operations involved in the simulation is parallelizable, the workload can be scheduled onto separate computational units to be evaluated at the same time. 10x to 100x acceleration can be achieved through replacing a serialized code with its parallelized counterpart [6].

1.2. The Landau-Lifshitz-Gilbert Equation

The LLG equation defines the precessional and damping dynamics of the magnetization $\mathbf{M}$. The description of the time evolution of the magnetization was first proposed by Lev Landau and Evgeny Lifshitz in the form of the Landau-Lifshitz equation [7]:

$$\frac{d\mathbf{M}}{dt} = -\gamma \mathbf{M} \times \mathbf{H}_{\mathbf{eff}} - \alpha \frac{\gamma}{M_s} \mathbf{M} \times \mathbf{M} \times \mathbf{H}_{\mathbf{eff}}, \tag{1.1}$$

where γ is the electron gyromagnetic ratio, and α is a phenomenological damping parameter. The component $\mathbf{H}_{\mathbf{eff}}$ is the effective magnetic field that the magnetization is exposed to, which is the combination of the magnetostatic field, exchange field, anisotropy field, external field, spin-transfer torque, magneto-restriction effect, Dzyaloshinskii-Moriya interaction (DMI) effect and etc. The evaluation of the effective field is among the most important components of solving the LLG equation, which will be discussed in detail in the following sections.

The first term in Eq. (1.1) is the precessional term. The magnetization is driven by the torque generated by the effective field to precess and the magnetization precesses along the axis defined by the effective field if the second term of Eq. (1.1) is zero. The second term is called the damping term. The magnetic system goes to the equilibrium due

to the energy dissipated described via the damping term. Note that even though the system energy changes due to the damping term, the magnitude of the magnetization vector preserves its magnitude. Combining the two terms, the magnetization in a magnetic system generally goes to the equilibrium state in a precessional manner.

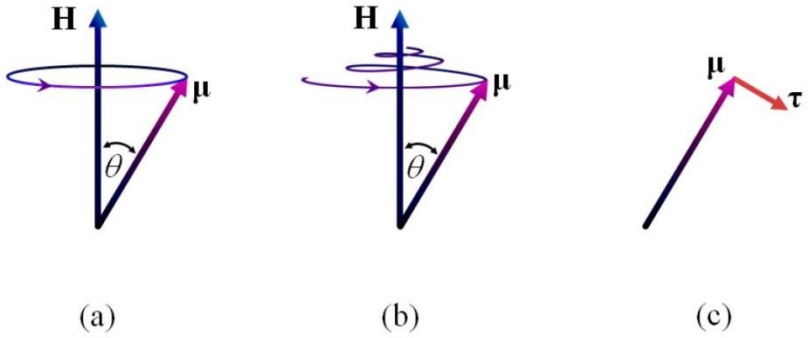

Figure 1.1: (a) Magnetization processing around the effective field; (b) Magnetization processing around the effective field with damping effect; (c) Action of arbitrary torque on the magnetization [3].

An alternative representation of the Landau-Lifshitz equation was given by T.L. Gilbert [8] and is referred to as the Landau-Lifshitz-Gilbert (LLG) equation

$$\frac{d\mathbf{M}}{dt} = -\frac{\gamma}{1+\alpha'^2}\mathbf{M}\times\mathbf{H}_{\mathbf{eff}} - \frac{\alpha'}{M_s}\mathbf{M}\times\frac{d\mathbf{M}}{dt}\ , \tag{1.2}$$

where α' is a phenomenological damping parameter which is different from α in Eq. (1.1), $\alpha' = \gamma\eta M_s$ where η is a material property related damping parameter.

The Eq. (1.2) can be re-written into

$$\frac{d\mathbf{M}}{dt} = -\frac{\gamma}{1+\alpha'^2}\mathbf{M}\times\mathbf{H}_{eff} - \frac{1}{M_s}\frac{\gamma\alpha'}{1+\alpha'^2}\mathbf{M}\times\mathbf{M}\times\mathbf{H}_{eff}\ . \tag{1.3}$$

This equation form is most prevalently utilized in the micromagnetic world and it will be used in this dissertation. Note that although Eq. (1.1) and Eq. (1.3) are the same in

terms of the format of having both $\mathbf{M}\times\mathbf{H}_{eff}$ and $\mathbf{M}\times\mathbf{M}\times\mathbf{H}_{eff}$ terms, they are different equations. The difference lies in the coefficient. For example, the coefficient of the precessional term $\mathbf{M}\times\mathbf{H}_{eff}$ depends on the damping in Eq. (1.3).

1.3. Governing Micromagnetic Interactions

The behavior of the magnetization **M** in LLG Eq. (1.3) is determined by the effective field $\mathbf{H}_{\mathbf{eff}}$, which includes components arising from several interactions. Among these interactions, the magnetostatic interaction, magneto-crystalline interaction, exchange interaction, and Zeeman interactions are the four typical components. Apart from them, magnetostriction, spin transfer torque, and Dzyaloshinskii-Moriya Interactions (DMI) may have a strong impact on the magnetization behavior. We review these interactions next.

1.3.1. Magnetostatic Interaction

A magnetic dipole moment generates a magnetic field with closed field lines, as shown in Fig. 1.2.

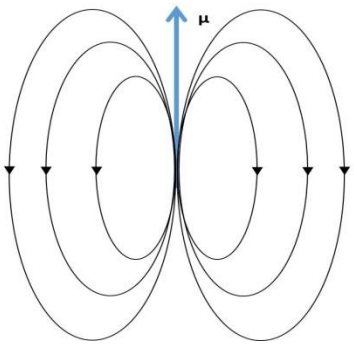

Figure 1.2: A magnetic dipole moment generates magnetic fields that form closure loops.

When multiple magnetic moments presents, they interact with each other through the magnetostatic field. The magnetostatic energy takes the form

$$E_{ms} = -\frac{1}{2}\int_V \mathbf{M}(\mathbf{r})\cdot\mathbf{H}_{ms}(\mathbf{r})d^3\mathbf{r}\,, \tag{1.4}$$

where the magnetostatic field $\mathbf{H}_{ms}$ can be defined as superposition integral:

$$\mathbf{H}_{\mathrm{ms}}(r) = \nabla\nabla\iiint_V \frac{\mathbf{M}(\mathbf{r}')}{4\pi|\mathbf{r}-\mathbf{r}'|}dV' \,, \tag{1.5}$$

which has a double derivative operator outside the integral. Alternatively, moving the rightmost differential operator under the integrand (via integration by parts) and defining effective volume and surface charge densities

$$\rho_M(\mathbf{r}) = -\nabla\cdot\mathbf{M}(\mathbf{r});\ \sigma_M = \hat{\mathbf{n}}\cdot\mathbf{M}(\mathbf{r}) \,, \tag{1.6}$$

the magnetostatic field can be given by its volume and surface integral components:

$$\mathbf{H}_{ms}(\mathbf{r}) = -\nabla\iiint_V \frac{\rho_M(\mathbf{r}')}{4\pi|\mathbf{r}-\mathbf{r}'|}dV' - \nabla\oiint_S \frac{\sigma_M(\mathbf{r}')}{4\pi|\mathbf{r}-\mathbf{r}'|}dS' \,. \tag{1.7}$$

Although no physical magnetic monopoles or magnetic charges exist, this form of expression gives an analogy to the electric field generated by electric charge densities. As a result, Eq. (1.7) gives a convenient way of computing and understanding the magnetostatic field.

The magnetostatic field has a significant effect on the magnetization dynamic and static behavior. For example, the magnetization in a magnetic structure tends to end up in an equilibrium state that minimizes the energy. In cases, where the magnetostatic field is dominant, the minimization is mostly related to the minimization of the magnetostatic energy. An example is a vortex state formed in a magnetic cube of a sufficiently large

size. Figure 1.3 shows a so-called μMag standard problem 3 [9] result. In this case, the magnetic interactions lead to the magnetization of a vortex state.

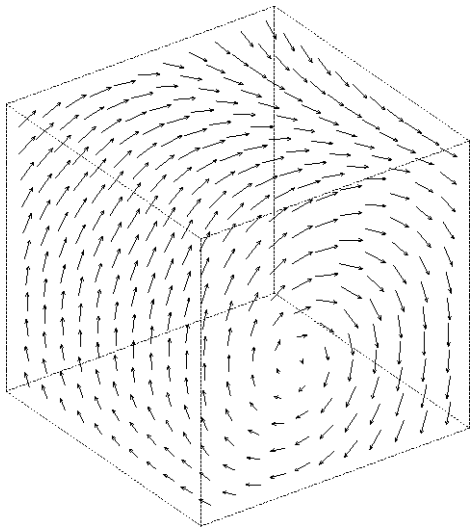

Figure 1.3: Vortex state formed in the magnetic cube [9].

1.3.2. Magnetocrystalline Interaction

Magnetocrystalline interaction reflects the symmetry of the crystal structure in a magnetic structure. For example, often existing magnetocrystalline interaction is uniaxial anisotropy, which leads to a preferential magnetization direction along the so-called easy axis. The uniaxial anisotropy energy is minimized when the magnetic moments align with the easy axis. The energy is given by the following expression:

$$E_{anis} = -\int_V K(\mathbf{m}\cdot\mathbf{k})^2 d^3\mathbf{r}\,, \tag{1.8}$$

where **m** is the unit magnetization vector, and **k** is the uniaxial direction. In this formulation, *K* is positive, meaning that the magnetization tends to align with the easy axis **k** . When K is negative, then the magnetization prefers to stay perpendicular to **k** , in which case **k** is called hard axis.

The magnetocrystalline interactions can also be written as a part of the effective magnetic field via

$$\mathbf{H}_{anis} = \frac{2K}{M_S}(\mathbf{m}\cdot\mathbf{k})\mathbf{k}, \tag{1.9}$$

Cubic anisotropy and higher order terms also may be important [10].

1.3.3. Exchange Interaction

Exchange interaction resembles springs between the magnetic moments to bond the moment vectors in the same direction. The exchange energy can be defined as

$$E_{ex} = -\int_V A\nabla^2\mathbf{m}(\mathbf{r})d^3\mathbf{r}, \tag{1.10}$$

where A is the exchange constant Taking the derivative of the energy over the magnetization vector, the expression for the exchange field can be obtained as

$$\mathbf{H}_{\text{exch}} = \frac{2A}{\mu_0 M_S}\nabla^2\mathbf{m}, \tag{1.11}$$

where M_S is the saturation magnetization.

Eq. (1.10) tells that exchange interaction prefers the uniformity of the magnetic moments. However, the magnetostatic interaction promotes the magnetic moments following the axis of the magnetic samples. Magnetocrystalline interaction prefers the moments to stay align with the easy axis so that the magnetization is such as it minimizes the overall energy. Such competition between the exchange interaction and other magnetic interactions may generate a variety of magnetization configurations, including uniform, domain wall, vortex, and other configurations.

1.3.4. Zeeman Interaction

The Zeeman interaction is the interaction between the magnetization and the applied/external field. The energy related to the Zeeman interaction can be expressed in the form

$$E_z = -\int_V \mathbf{M}(\mathbf{r}) \cdot \mathbf{H}_z(\mathbf{r}) d^3\mathbf{r} \tag{1.12}$$

It is obvious from Eq. (1.12) that the magnetization $\mathbf{M}$ tends to align in the same direction as applied/external field $\mathbf{H}_z$ to obtain the lowest energy state. On the other hand, it maximizes the Zeeman energy if the magnetization and the applied/external field $\mathbf{H}_z$ lie in the opposite directions.

Zeeman interaction is widely exploited in the magnetic systems in the real world. For example, the magnetostatic field generated by the magnetic write head can be taken as an applied/external field in the magnetic media thin films to manipulate the magnetization orientation in the thin films. The hard-drives utilize this effect to record the binary data.

1.3.5. Other Interactions

STT effect depicts the phenomenon that lattice is capable of absorbing the angular momentum from the spin-polarized current. This effect has promising applications because it allows manipulating the magnetization of a certain magnetic sample by applying a local spin-polarized current in the system. Typical applications of the STT effect are Magnetic RAM (MRAM) and spin-torque oscillators (STO). Note that STT is

an effect not directly related to the energy, so it does not contribute to the total energy of the magnetic system.

Dzyaloshinskii-Moriya Interaction (DMI) is an antisymmetric exchange coupling [11]. It arises from a broken symmetry in the system, which could be realized at surfaces of thin films. DMI is drawing a great interest of the magnetism community since it could induce chiral spin structures such as skyrmions [12] and other unconventional phenomena [13] [14] [15]. Fig. 1.4 shows skyrmion simulated by the numerical magnetic solver FastMag. The magnetization is pointing up on the edges and pointing down in the center [12]. The technology could be used to produce new-generation of spintronic devices.

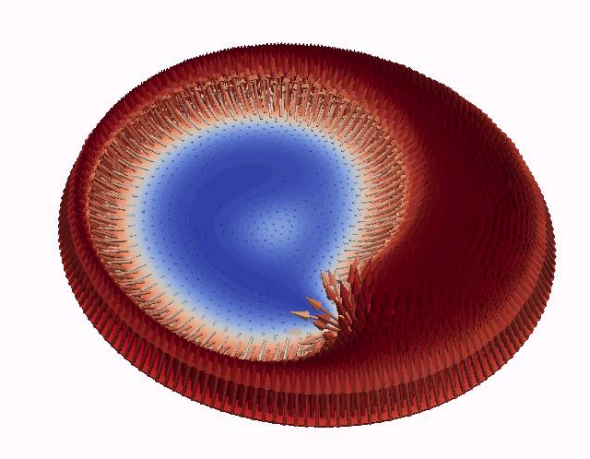

Figure 1.4: Skyrmion simulated by FastMag.

1.4. Finite Difference Methods and Finite Element Methods

The two main types of micromagnetic solvers are based on the Finite Element method (FEM) [16][17][18][19] and Finite Difference method (FDM) [20][21] [22]. The

two methods are different in how the structures are discretized. As a result, their formulation, implementation, speed, and numerical accuracy are different.

FDM uses a regular grid of rectangular brick cells, at which the differential operators can be approximated by central differences. The size of each brick cells is $\Delta x \times \Delta y \times \Delta z$, and the time step is Δt. For explicit time integration schemes, to maintain numerical stability, the time step Δt has to decrease with finer space discretization (Δx, Δy, Δz), as what the Courant–Friedrichs–Lewy condition states [23].

Due to the regularity of the discretization grid, the formulation of the micromagnetic modeling with FDM is relatively simple and the implementation is quite straightforward. Moreover, the computational speed of FDM can be good for simple magnetic structures, such as rectangular thin films. Therefore, it is extensively utilized in the micromagnetics community for such cases. On the other hand, FDM suffers from certain factors that prevent a universal application of the method. Most importantly, the modeling accuracy for the magnetic samples that come with fine geometrical features can be unsatisfactory. This is due to the fact that regular brick cells intrinsically are not well suited to model curved boundaries.

FEM greatly solves the problem by applying arbitrary shaped finite elements in the mesh. Each finite element could be a triangle, a quadrilateral, or even a curved triangle in 2-dimensional case. As to 3-dimensional mesh, the elements could be tetrahedrons, hexahedrons, pyramids and prisms [24]. The flexibility in the discretization gives the better geometric modeling accuracy. The modeling flexibility and accuracy come, however, with complexities in formulation and implementation. Since sparse matrix inversion is often involved in FEM, the computational speed may be slower than

that of FDM when handling simple magnetic structures. To efficiently invert the matrix, a suitable preconditioner may be necessary to accelerate the process. Preconditioning will be discussed in Chapter 4.

Both FDM and FEM are successfully used in the micromagnetics community. The formulations, implementations, and applications of both methods are discussed in this dissertation. The choice of numerical modeling method depends on the application. A rule of thumb is for simple magnetic structures, such as rectangular thin films, one may choose FDM while for complicated structures, such as magnetic write heads or multilayered MRAM cells, one may choose FEM.

1.5. FastMag: A Fast and Accurate Micromagnetic Solver

FastMag is a FEM based micromagnetic simulator that is intended for solving problems of high geometrical and material complexity. FastMag's flexibility and performance rely on several advanced computational methods, including fast evaluation of effective fields, fast evaluation of the system Jacobian, efficient time integration, efficient methods for system discretization, and the ability to run on various computing platforms [6]. A brief description of the FastMag computational components and the outline of the performance and optimization of these components are given here.

The main components of the FEM-based micromagnetic codes of FastMag include the evaluation of the magnetostatic field and exchange field, computing the system Jacobian, and time integration. From the computational mathematics point of view, these components can be cast in terms of dense products, sparse products, FFTs, and iterative solvers using these products.

In FastMag the evaluation of the magnetostatic field is accomplished by computing the superposition integrals based on the magnetization states at the nodes of a tetrahedral mesh, which is different from a more conventionally used scheme in micromagnetics based on solving the Poisson equation. The evaluation of the integrals is based on the assumption of a constant of linearly varying magnetization in each tetrahedron, which is obtained from its values at the nodes. The computation of the magnetostatic field includes several steps:

- For each node (i.e. a vertex of a tetrahedron), near- and far-field nodes are identified based on their distance, e.g. with respect to the largest edge in the mesh.
- The fields at the near-field nodes are computed directly via analytical integrals. From the code implementation point of view, this step includes a tabulation of the field interactions, which is a sparse matrix representation of the field contributions to each node from its near-field node list. These computations are accomplished via sparse matrix-vector products.
- For computing far fields, the magnetic scalar potential is first found via a quadrature rule. The quadrature rule translates the integrals into discrete summations. The quadrature rule effectively translates the vector magnetization states into scalar charges assigned to the mesh nodes (or to the quadrature points) and converts the continuous integrals into discrete sums.
- The magnetic scalar potentials at the far-field nodes are computed via the non-uniform Fourier transform (NUFFT), which also is referred to as Adaptive Integral Method in Electromagnetic solvers [25]. NUFFT includes four main steps: Projection of the spatially non-uniform nodal charges into a uniform grid;

Convolution computation of the potentials on the uniform grid via FFT; Interpolation of the potential from the uniform grid onto the non-uniform mesh node locations; Corrections for the closely located points by using direct summations. Some of these steps are cast into dense and sparse matrix-vector summations and some the steps are computed on-the-fly without defining any matrix operations.

- Once the potential values are found, the magnetostatic field is computed as the divergence. This operation is cast into a sparse matrix-vector product.

The computation of the exchange field is accomplished using the weak FEM formulation with the box-method [26]. In this approach, the field at each mesh node is obtained by tabulating a sparse stiffness matrix, which gives the exchange contribution at each mesh node from its surrounding nodes. The evaluation of the exchange field involves a sparse matrix-vector product each time exchange field is needed. The time integration of the LLG equation in FastMag can be accomplished via the implicit multi-step backward differentiation formula (BDF) [27] [28]. For stiff problems, which often appear in micromagnetics, the time step required for numerical stability can be very small, which can make simulations slow. Implicit time integrators methods can significantly increase the time steps, thus increasing the simulation speed and code robustness. The implicit implementations require solving a non-linear system at each time integration, usually via Newton iterations. At each non-linear iteration, a linear system is solved via an iterative solver such as GMRES (direct inversion solvers also can be used but only for small problems). This iterative process requires evaluating the system Jacobian. FastMag implements an approach in which the system Jacobian computation is entirely based on the computation of the effective fields used together with analytical formulas. Therefore,

the computation of the Jacobian involves the same operations as those required for the effective fields.

Chapter 1, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, R. Chang, S. Couture, M. Menarini, M.A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay, V. Lomakin, "*Micromagnetics on High-performance Workstation and Mobile Computational Platforms*". Journal of Applied Physics, vol. 117, no. 17, pp. 17E517, 2015.

2. Parallel Computation on GPU

The speed limitation of single-core systems has become an obstacle when solving large-scale problems in micromagnetics and other fields of study. Micromagnetic solvers for multicore and multi-CPU computing systems have been developed, but such systems have limitations in their performance. Massively parallel GPU computer systems have emerged offering ultra-high performance. A single GPU can match the computation power of a middle-range CPU cluster, but at a much lower cost and power consumption. Solvers for the LLG equation on GPUs can be highly efficient but several important points need be addressed to fully exploit the computational power of GPU architectures [6] [16][21][29].

In this chapter, we describe parallel computing applied to GPUs. The programming models on GPUs may be different from those on CPUs, and we introduce a few important programming points to achieve an optimized performance. Moreover, use of GPUs on various platforms such as desktops, servers, and embedded systems are reviewed. Finally, we conclude this chapter with uses of GPUs in the micromagnetic community.

2.1. Introduction to GPU Computing

GPUs are one of the many processor types in a modern desktop, laptop, clusters and embedded systems. The massively parallel system GPU was originally designed for graphics related applications, such as display, image, video, gaming, etc. The focus of the

GPU is the data throughput, namely processing a large amount of data with the same operation pattern at the same time. However, the impacts of GPUs have stepped beyond its original scope. As a result, a new term, General Purpose GPUs (GPGPUs), is becoming increasingly popular in a number of computational communities. Computational scientists and researchers put much effort in designing or modifying their data or compute intensive algorithms to run on GPUs so that 10x – 1000x acceleration could be attained [30].

Figure 2.1: One of the latest Nvidia GPUs, GTX 1080.

Many-core systems such as GPUs are well suited for parallel computing, where a computational job is divided into a number of independent but similar operations. We note that the benefit of GPU is not universal since there are certain kinds of applications that have to stay with serial operations, as shown in Fig. 2.2 (a). Specifically, many-core systems are efficient for applications that are similar to the one shown in Fig. 2.2 (b).

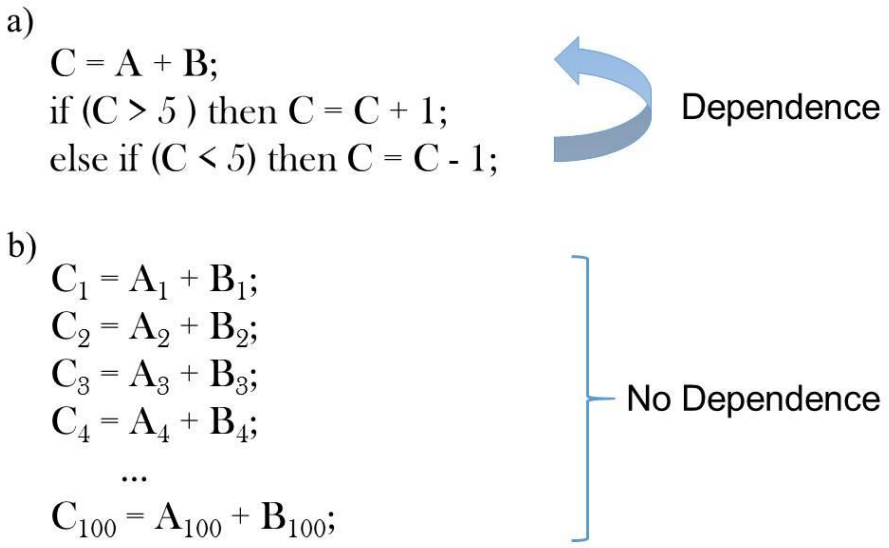

Figure 2.2: (a) Serial application vs. (b) parallel applications. The dependence between operations in (a) closes the opportunity for parallelization.

Many real world applications are parallel-friendly. For example, the n-body problem evaluates the gravity between moving celestial bodies, as a result, the computational complexity is $O(N^2)$. However, since the gravity only depends on the mass and the location of the interacting two bodies, regardless of all the other N – 2 bodies in the simulation, the compute-intensive operations in this problem are independent. The GPU acceleration can be best demonstrated on such application types [31]. It also can be efficient for fast methods that scale computationally as $O(N)$ or $O(NlogN)$.

2.2. GPU Programming Model

Through the device driver and programming language, developers are able to focus on the computational operations instead of the hardware details. As a result, the differences of the hardware implementation details from various vendors are hidden.

OpenCL and CUDA are the two mainstream programming languages for GPUs. OpenCL can run on GPUs from both mainstream vendors AMD and Nvidia, while CUDA can only run on Nvidia GPU but it was the first and most common GPU programming approach. In this dissertation, CUDA is the focus.

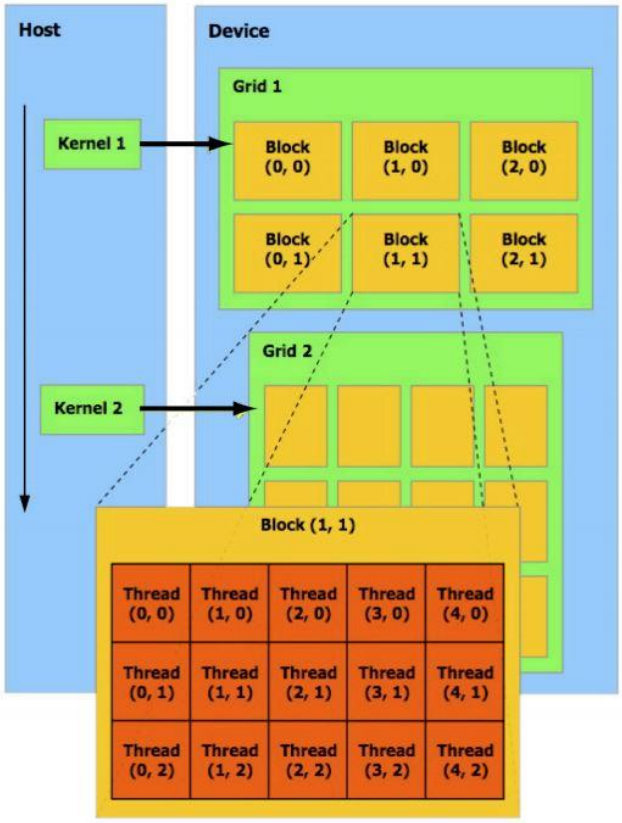

Figure 2.3: CUDA programming model.

With CUDA it is straightforward to control the stream processors (SPs) and multiple levels of memory as well as memory transfers between CPU and GPU. In the context of CUDA, the computational resources are organized in three levels: grid, blocks, and threads (Fig. 2.3). If we name the subroutines that run on GPUs as kernels, then each kernel has a grid, which is composed of a few blocks (e.g. 6 blocks). Inside each block,

there are a few threads (e.g. 15 threads). A thread is a set of instructions to be executed sequentially by a GPU SP. However, it is easier to understand by regard a thread as the smallest launched instances in CUDA, and multiple threads could run in parallel in the GPU. For example, let us consider a case in which there are $6 \times 15 = 90$ threads in parallel for Kernel 1. Each thread can be identified with a unique thread index. Similarly, each block comes with a unique block index. As a result, all the threads and blocks within a grid are indexed. Note that these are all software concepts, concretely, the total number of threads could be more than actual hardware SPs. In that case, the threads can be scheduled onto the SPs serially automatically.

Programming models for GPU can be compared with those for many-core CPUs. In CPUs, the computational tasks can be parallelized in a few task-paths onto different physical cores, as shown in Fig. 2.4(a). Each CPU core is powerful and the jobs can be handled at the same time. The Task-parallel model is efficient when handling different kinds of jobs even if they do not share the same computational pattern. On the other hand, the GPU uses Data-parallel model. Namely, a task is passed into a single operation scheduler so that the data could be divided into pieces for different cores, as shown in Fig. 2.4(b). Each GPU core is less powerful here, compared to a CPU core, but the throughput of a GPU can be much larger than a CPU due to a large number of cores.

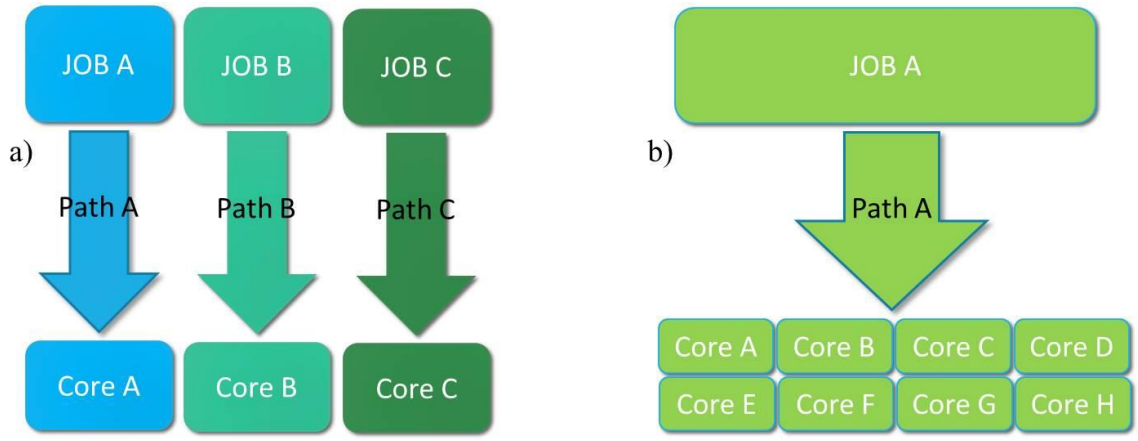

Figure 2.4: Programming models: (a) CPU programming model: task parallel; (b) GPU programming model: data parallel.

The programming model decides how the CUDA language is designed. As an extension to the "C" programming language, CUDA inherits many features from "C". Fig. 2.5 is a sample program to add two long vectors. On the host (CPU) side, a GPU kernel "vectorAdd_parallel" is called. Note that the grid structure and the block structures have to be specified in the brackets "<<<…>>>". In this example, there are 16 blocks in the grid and each block is a bundle of 192 threads. The GPU kernel "vectorAdd_parallel" is executed by every launched instance, namely, the $16 \times 192 = 3072$ threads. The index of the instance i is identified with "blockIdx", "blockDim" and "threadIdx" so that each thread could be assigned to a unique location in the arrays.

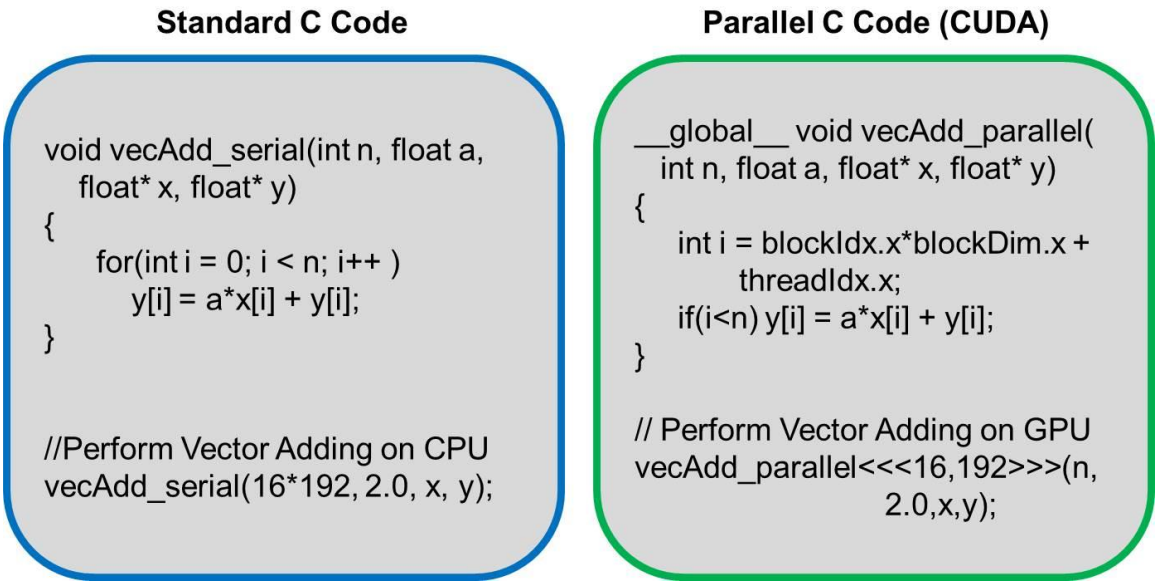

Figure 2.5: Comparison of a C program and a CUDA program working on vector adding.

2.3. GPU Programming Points

One of the latest consumer line GPUs NVIDIA GeForce GTX 1080 has 2560 stream processors. These processors are launched by CUDA threads in the CUDA programming environment [32]. In GTX 1080, 64 stream processors form a streaming multi-processor. All processors within a certain multi-processor share a certain amount of L1 cache and fast shared memory. There is also the GPU global memory, which is much larger but also slower. Threads within a GPU have access to global memory while threads within a block only have access to this block's shared memory. Unlike the cache, the shared memory on GPU is managed by the programmer. It is imperative to take advantage of the shared memory for developing highly efficient GPU code.

The number of "IF" statements should be minimized for speed consideration. Each stream processor can launch one thread at a time, 32 threads form a warp which is

an “atomic” execution unit. Here, “atomic” execution means all the 32 threads execute the same instruction at a given time. In other words, a warp is the smallest execution unit in GPU. The reason for this is the “atomic” execution behavior of a warp. For example, we can assume that it takes one time cycle for a certain warp to execute the same instruction. However, it may take several cycles if threads within a warp have different types of instructions. The need to execute different instruction types would require some threads to wait, thus decreasing the computational efficiency.

Occupancy is another important parameter to explore the power of GPU. To hide the latency of certain executions, a single streaming multi-processor (SM) holds many warps such that when some warps are temporarily stalled, the schedulers of SM can switch to other launch-ready warps. Therefore, the more active warps on an SM, the more latency is hidden. However, the number of warps controlled by a single SM is limited by the available resources on a single SM, such as registers, shared memory, and block sizes.

GPU global memory is a separate part of memory from CPU memory, so any data that needs to be operated on GPU has to be transferred from the CPU memory. The GPU global memory has significant access latency. This latency is hidden when reading data from the continuous addressed in the memory via so-called coalesced access. It is important to arrange data uniformly in the memory so that coalesced access mechanism can be utilized.

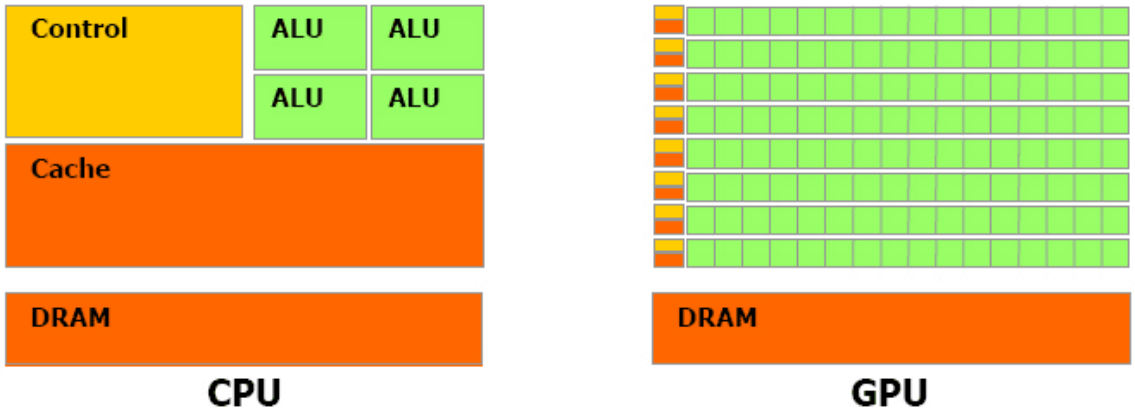

Figure 2.6: CPU and GPU brief-architecture: the CPU and GPU have separate memories. CPU focus more on powerful ALUs, while GPU focuses more on massive data processing.

GPU also has a category of "shared memory". Shared memory is very fast on-chip memory and it works as a memory pool for the threads to share the intermediate data. Synchronization may be launched to avoid the race condition. The maximum size of shared memory on a single SM is 48KB. On the other hand, Nvidia Pascal architecture provides another way to share data in registers between threads within a warp: the shuffle instruction. Shuffle instructions allow exchanging data in a warp of threads by enabling the threads to access other threads' registers. Shuffle instructions can be used in the operations including reductions, scans, transposition and sorting. It has been proved to be always faster than the safe shared memory operations [33]. Since the space of shared memory is limited, shuffle instruction is favorable for certain applications that lack shared memory resource.

Although shared memory is as fast as cache, its access can be slowed down by bank conflicts. In Nvidia GPU, the shared memory is organized into memory banks, and memory locations on the same bank can only be accessed once at a time. Thus, if there

are two threads within one warp trying to access the same bank, these two accesses have to be serialized. To avoid this, a linear accessing stride is preferred.

2.4. GPUs on Various Platforms

Multiple parallel platforms have been used in micromagnetics, including multi-core central processing units (CPU) and graphics processing units (GPU). In particular, GPUs were introduced offering ultra-high performance, which allowed using inexpensive desktop computers as high-performance computer clusters. Recently, new embedded mobile-based computer architectures emerged, and their performance increase has been higher than that of desktop CPU and GPU systems. In addition, embedded systems offer low power consumption and manufacturing cost [6].

In this section, we explore using micromagnetic solvers on high-performance parallel platforms, including desktop multi-core CPU, desktop GPU and embedded mobile computing architectures. The feasibility and convenience of using these systems for high-performance computing will be investigated.

2.4.1. GPU in Desktops and Laptops

Traditionally, all the desktops and laptops that are equipped with separate GPUs were designed for gaming or graphics-related work. Ever since GPGPU became practical in 2001, it has attained tremendous achievement in the compute intensive applications such as computational photography and physics simulations. This wide adoption of GPGPUs has reversely encouraged the vendors of GPUs to improve their architecture further for computing tasks. Apart from the companies AMD and Nvidia that ships GPUs,

Intel has commercialized a relatively new many-core system called Xeon Phi [34]. The GPUs from AMD and Nvidia currently are dominating the market, while Xeon Phi is appearing to be a strong competitor.

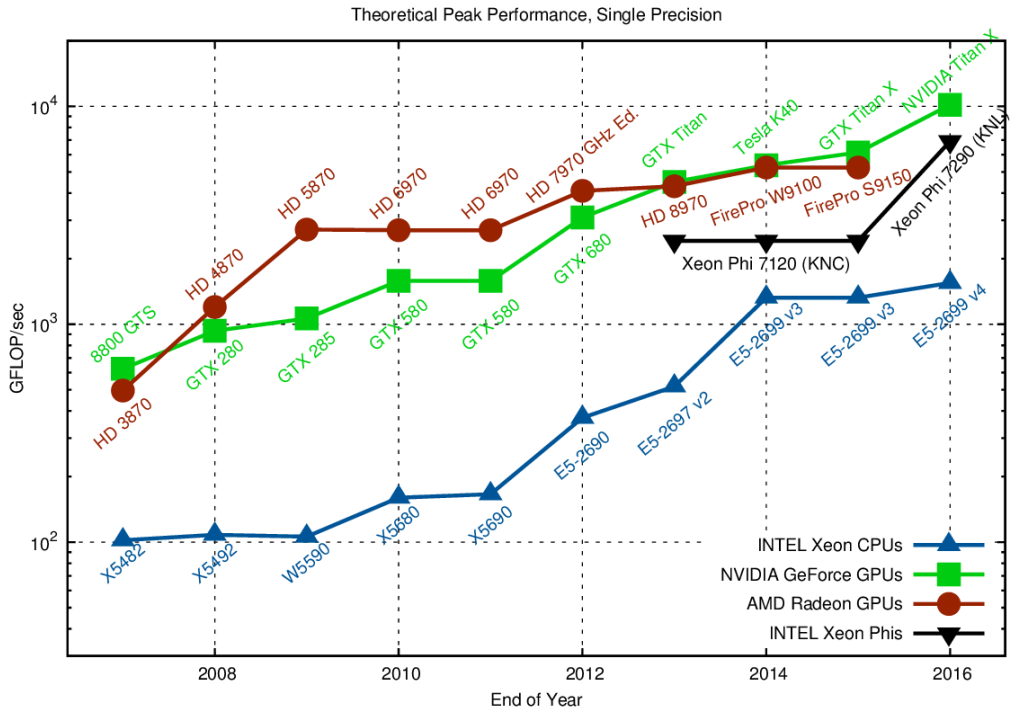

Figure 2.7: Theoretical peak performance in single precision. (The data for 2016 is preliminary.) [35].

The computational power of GPGPUs has been increasing rapidly over the recent years, as indicated by the theoretical peak performance data in Fig. 2.7. The raw computational power of the strongest GPU is about 10x over the latest version of CPU. The increase of the power of GPUs has outpaced that of CPUs. As discussed in previous sections, the strong computational power of GPU relies on its capability on handling a large amount of data at the same time. This ability cannot be real without large memory bandwidth, which guarantees the availability of data when the streaming processors are ready to process it.

Utilizing GPGPUs on the desktop for computational purpose effectively reduces the cost of High-Performance Computing. It allows running real-world complex large-scale simulations on normal computers. The price of one of the latest GPGPUs from Nvidia GTX 1080 is only $699, with a raw performance of 9 GFLOP/s and 8 GB memory. Nowadays, the GPGPUs on desktop serve as an important resource of scientific computation. GPGPU is a reasonable start point of utilizing GPU for scientific research, and then high-end cluster GPU could be introduced to further improve the performance.

2.4.2. GPUs in Servers/Clouds

An emerging computational power is the cloud computing which quickly dominates the commercial computing field. Researchers prefer the centralized computational resources to accelerate their applications with the controllable budget. Cloud computing's relatively low cost allows the researchers to exploit its full power from weather prediction to molecular dynamics. Cloud computing provides massive scalability to allow ultra-scale computation with very high performance, which is essential for the resource-hungry applications. Also, the clouds hide the layer of infrastructure from the users to free the commercial users and researchers from the maintenance work. The servers provided in the cloud computing service are maintained by the computer architecture experts, the backup service scheme provides an almost 100% reliability to the normal users. Here, the reliability comes in three folds. Firstly, the data is guaranteed to be secure since there is redundant data storage. Secondly, the service will not fail easily because of a single point since there are always redundant resources ready to be utilized. Thirdly, each cloud user is isolated to avoid cyber-security problems.

Moreover, the users are able to specify the demands so as to avoid wasting the computing resources [36]. It is not hard to predict that the computation executed in the clouds are much more energy efficient than the individual desktops.

A famous application of the GPUs in large-scale clusters and clouds are the deep learning in the context of the artificial intelligence. In deep learning, a neural network like computing model is constructed to divide the learning process into a few layers that are composed of nodes (neurons). In general, the prediction accuracy improves with the number of nodes and layers involved, however, this is at the price of the more complicated model and computational time. In 2016, a deep learning cluster model by Google, named "Alpha Go" beat the world's best Go player [37]. Go is said to be one of the most complicated game in the world, way beyond the chess game [38]. The ability to compute the optimized strategy relies on the high computational power delivered by the GPU clusters.

Figure 2.8: GPUs in the cloud computing [39].

The market for cloud computing is growing rapidly, including the computation and graphics rendering work delivered by GPUs. Scientists can now process petabytes of data up to 10x faster than with CPUs in applications ranging from energy exploration to deep learning [40]. A series of GPU called Nvidia Tesla GPUs are delivered for scientific computing specifically. These GPUs are equipped with error-correction mode to guarantee the accuracy of the results computed by the GPU, which is different from the GPGPU. GPGPU may deliver relatively high errors due to the fact that they are designed for graphical rendering which does not require high accuracy. The Tesla GPUs are widely used in, but not limited to, the scientific computer centers. Researchers have investigated the power of cloud computing for the micromagnetic simulations [41]. It is promising that the cloud computing could be the dominant resource for the micromagnetic simulations so that the benefits of cost effectiveness, strong computational power, scalability, and reliability could be exploited.

2.4.3. GPUs in Embedded Systems

Recently, new embedded mobile-based computer architectures emerged, and their performance increase has been faster than that of desktop CPU and GPU systems. Embedded systems have been dramatically changing our daily life since the mobile devices such as smartphones and pads grow its popularity. In these devices, usually, a GPU is integrated with a multi-core CPU on the same chip to render the graphics. In addition, embedded systems offer low power consumption and manufacturing cost. Therefore, there are chances that researchers could utilize the computational power out of the mobile GPUs.

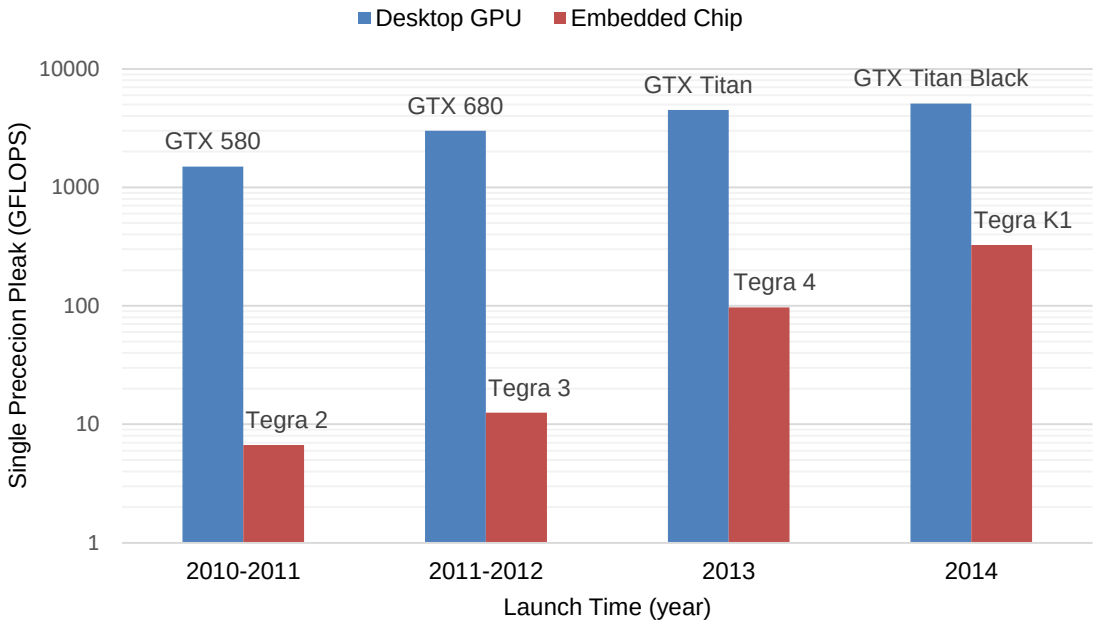

Figure 2.9: Computational power comparison between desktop GPUs and embedded chips [6]. All chips shown are from Nvidia, and dual GPUs are not included for fair comparison. Data collected from open-source benchmarks [42].

Fig. 2.9 demonstrates that performance increase of mobile systems currently outpaces that of traditional GPUs and they also have a significantly lower power envelope. Jetson TK1 was the first embedded mobile supercomputer that enables fully functional Nvidia CUDA, which allows porting micromagnetic codes developed for desktop GPUs. Jetson TK1 is a miniature-sized system that has all computer components, including a motherboard, mobile CPU, mobile GPU, memory, and storage space. Jetson TK1 delivers high computational performance (326 GFlops) at a low cost ($192 for the entire system) and low power consumption (10w). Jetson TK1 is powered by Nvidia Tegra K1, which breaks the performance record of embedded system chips and can be used in any mobile platforms such as mobile phones and tablets. Tegra K1 has 192 GPU cores in a single streaming multiprocessor. As a comparison, the currently one of the

most powerful desktop GPUs, Nvidia GTX 1080, has 2560 GPU cores in 40 streaming multiprocessors with 180w power consumption, around $700 cost, and it requires a host computing system with a separate CPU, motherboard, memory, and storage drive. Tegra K1 and GTX 1080 share a similar, but not the same, GPU architecture, which makes their coding similar, with some optimization and compilation differences.

Considering that Tegra K1 GPU has 14x fewer GPU cores than, e.g., GTX 1080, the Jetson parallelism saturates faster, which means that Tegra K1 can be more beneficial for smaller problems. Jetson TK1's memory has a lower bandwidth so that Tegra K1's performance is more sensitive to the memory access. The GPU-CPU memory transfer is also slower on Tegra K1 (10x slower from CPU to GPU and 2x slower from GPU to CPU), which means that applications that offload all computations to GPU would benefit more on Jetson TK1.

With Linux operating system pre-installed, Jetson TK1 is a platform ready for the massive computational workload. The network port allows Jetson TK1 connecting with other devices, implying the possibility to build cluster at a low cost and power consumption.

2.4.4. Numerical Results

As outlined in Chapter 1 the numerical operations required in the FEM micromagnetic code, such as FastMag, include dense products, sparse products, and NUFFT with FFTs.

Since Tegra K1 has a similar GPU architecture as desktop GPUs, the optimization methods are similar as well, but there are also some differences. The GPU of Tegra K1

has less shared and register memory per core so that Tegra K1 GPU performs better with compute-intensive applications. In particular, it requires using a large near-field size in the NUFFT optimization. The shared and register memory size limitation may lead to occupancy issues for certain GPU kernels that run well on desktop GPU. Therefore, splitting long GPU kernels into shorter ones may be beneficial on Tegra K1. Tegra K1 has lower memory bus width, slower memory clock rate, and smaller L2 cache so that applications sensitive to memory access speeds should be careful in coalescing the GPU global memory access and taking advantage of shared memory and warp-shuffle instructions to minimize the global memory traffic.

Table 2.1: Performance of desktop CPU, GPU and embedded GPU, with speed-up against desktop CPU with a single core.

	Problem Size	CPU 1 core	CPU 4 cores	Jetson TK1	GTX 690
N-body	32K	9052 ms (1.0x)	2142 ms (4.2x)	137 ms (66.1x)	19 ms (484x)
	1K	9.39 ms (1.0x)	3.19 ms (2.9x)	1.48 ms (6.3x)	0.78 ms (12x)
NUFFT	270K	1924 ms (1.0x)	887.1 ms (2.2x)	341.8 ms (5.6x)	43.6 ms (44x)
	4K	13.0 ms (1.0x)	4.86 ms (2.7x)	8.94 ms (1.5x)	2.74 ms (5x)
FFT	2M	34.0 ms (1.0x)	15.7 ms (2.2x)	22.4 ms (1.5x)	1.5 ms (23x)
	4K	0.080 ms (1.0x)	0.037 ms (2.2x)	0.527 ms (0.2x)	0.093 ms (1x)
SpMVM	600K	17.09 ms (1.0x)	7.70 ms (2.2x)	17.4 ms (1.0x)	1.88 ms (9x)
	4K	0.074 ms (1.0x)	0.019 ms (3.9x)	0.60 ms (0.1x)	0.33 ms (.2x)

Table 2.1 shows the performance comparison between a 4-core CPU (Intel i7-3770), a desktop GPU (GTX 690) and an embedded GPU (Tegra K1). The N-body problem is a sample code by Nvidia, which provides both CPU and GPU benchmarks. NUFFT and SpMVM are implemented in FastMag, the details of which will be discussed in the following chapters. FFT is from Nvidia cuFFT library and it is applied in the NUFFT implementation. CPU results of FFT and SpMVM are from Intel MKL. Small

and large problem sizes are chosen for each algorithm. It is evident that the performance gap between Jetson TK1 and desktop GPU is significantly narrower for smaller problem sizes, which is due to the fact that the full utilization of parallelization on the desktop GPU is achieved at larger problem sizes.

The multi-core CPU results were obtained using OpenMP. Comparing the results of GPUs with the multi-core CPU, in general, the GPUs are favorable for large and compute-intensive applications. Jetson TK1 GPU is faster than multi-core CPU for n-body and NUFFT algorithms, while GTX 690 is significantly faster than multi-core CPU in most cases.

A Jetson TK1 CPU to Intel i7-3770 CPU comparison test was also conducted, showing that embedded CPU is 11.0x slower than desktop CPU. This ratio is larger than mobile GPU to desktop GPU difference, which means that using the GPU of Tegra K1 for computations is most beneficial.

To demonstrate the feasibility of using desktop and mobile CPU and GPU systems for micromagnetic modeling, Table 2.2 shows the simulation results of the FastMag simulator. The numerical tests included the time relaxation dynamics in a soft magnetic cylinder of sizes ranging from 300 nm radius by 50 nm height to 60 nm radius by 10 nm height. In order to evaluate the speed scaling on different computational platforms, we tested 7 different cases listed in Table 2.2. The geometry aspect ratios in all cases were kept roughly the same. The material properties were Ms=8.0e5 A/m, A=1.3e-11 J/m, K = 5.0e5 J/m3 and alpha=0.01. The cylinder was meshed with a tetrahedral mesh of around 4nm edge length, leading to the total number of nodes ranging from 3 thousand (0.01 million elements) to 300 thousand (1.5 million elements). The 300-

thousand node problem was the largest we could run on Jetson TK1 system, limited by the Jetson TK1 memory of 2GB. The GPU workstation we used for testing had a Nvidia GTX690 at 915 MHz (a single GPU device of the dual-GPU card was used), and one 4-core Intel-i7 3770 CPU at 3.4GHz. Each single GPU of GTX690 has 1536 cores. Jetson TK1 has 192 GPU cores at 852 MHz, while the integrated mobile CPU has 4 cores running at 2.3 GHz. Two time integration methods were tested: BDF for implicit time integration and Heun Method for explicit time integration. For all GPU tests, magnetostatic field, exchange field, and the numerical system Jacobian were evaluated on GPU. For the multi-core CPU tests, magnetostatic and exchange fields were computed on multiple cores, with 2 threads per core exploited. All results presented are from single-precision floating point operations.

Table 2.2: Performance of Various platforms and speed-up versus single-core CPU.

	#Nodes	#Elements	CPU 1 core	CPU 4 cores	GTX 690	Jetson TK1
Implicit Time Integration (mins/ns)	3K	0.01M	0.99 (1.0x)	0.67 (1.5x)	0.71 (1.4x)	2.9 (0.3x)
	10K	0.065M	12.4 (1.0x)	5.9 (2.1x)	1.7 (7.4x)	9.0 (1.4x)
	20K	0.1 M	30.1 (1.0x)	10.0 (3.0x)	2.7 (11.0x)	17.5 (1.7x)
	50K	0.25 M	115 (1.0x)	36.9 (3.1x)	7.7 (15.0x)	54.5 (2.1x)
	100K	0.5 M	186 (1.0x)	84.4 (2.2x)	18.5(10.1x)	136 (1.4x)
	170K	1.0 M	298 (1.0x)	126 (2.4x)	28.7(10.4x)	224 (1.3x)
	270K	1.5 M	539 (1.0x)	276 (2.0x)	61.1 (8.8x)	339 (1.6x)
Explicit Time Integration (ms/step)	3K	0.01M	18.9 (1.0x)	86.4 (2.2x)	5.87 (3.2x)	16.0 (1.2x)
	10K	0.065M	135 (1.0x)	52.5 (2.6x)	10.5(12.9x)	51.7 (2.6x)
	20K	0.1 M	228 (1.0x)	94.1 (2.4x)	16.0(14.2x)	109 (2.1x)
	50K	0.25 M	558 (1.0x)	217 (2.6x)	29.9(18.7x)	234 (2.4x)
	100K	0.5 M	1202 (1.0x)	471 (2.5x)	53.1(22.7x)	494 (2.4x)
	170K	1.0 M	2093 (1.0x)	912 (2.3x)	93.8(22.3x)	914 (2.3x)
	270K	1.5 M	3794 (1.0x)	1482 (2.6x)	172 (22.1x)	1456 (2.6x)

All the hardware types showed good performance. Parallelization of serial CPU code with 4 CPU cores achieved up to 3.1x acceleration. Jetson TK1 had up to 2.6x speed-up against a single core CPU and it was 3x-9x slower than the desktop GPU. The desktop GPU results were 3x-23x faster than the single core CPU results.

The implicit BDF time integration method, while being more efficient in time stepping as compared to the explicit time integration, had some performance limitations in terms of desktop GPU-CPU and Jetson TK1 - desktop CPU speed-ups. The limitation is related to the fact that in our implementation while most of the computationally intensive parts are on GPU a part of the BDF integrator is running on CPU, which restricts overall GPU-CPU gains. In particular, Jetson TK1 showed similar performance as multi-core CPU with explicit method while it was averagely 1.6x slower than multi-core CPU with the implicit method. Jetson TK1 CPU is 11.0x slower than the desktop CPU. A larger portion of workload on CPU with the implicit method led to less favorable results on Jetson TK1 platform. On the other hand, the explicit time integrator had a better GPU-CPU speed-up but the overall speed performance was slower due to the need for smaller time steps. We work to port the BDF time integrator to GPU, with expected additional performance gains.

We note that the raw simulation time performance is not the only metric when assessing the feasibility of using a computing system. Power consumption and cost are additional important metrics when assessing throughput of a computational facility, e.g. when one needs to run a large number of micromagnetic simulations for device design and optimization. The comparison among all tested platforms is summarized as Fig. 2.10, where Jetson TK1 platform showed an attractive operation in terms of the cost/power

consumption - performance ratio. When comparing Jetson TK1 to a desktop CPU system, the former is up to 2.6x faster than a single core, similar performance as multi-core, over 5x lower cost, and 20x-30x lower power consumption. When comparing Jetson TK1 to a desktop GPU system, the former was 3-9x slower but it is 10-20x less expensive and uses around 70x less power. Therefore, embedded mobile computing platforms have favorable cost and power efficiencies for micromagnetic simulations.

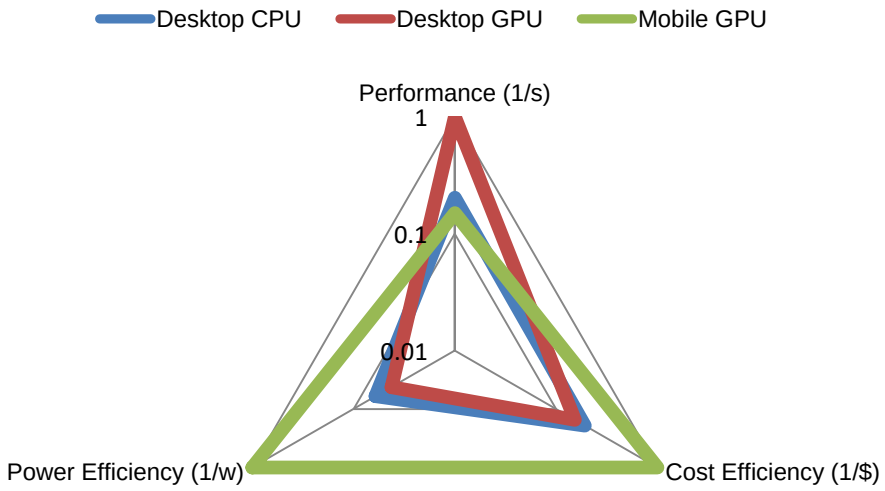

Figure 2.10: Comparison of performance, power efficiency and cost efficiency among desktop multi-core CPU, Desktop GPU (GTX 690) and Mobile GPU (Jetson TK1). The baselines of three criteria are normalized to 1. Performance results are based on micromagnetic simulations.

2.5. GPUs in Micromagnetics

Micromagnetic models based on the LLG equation and its derivatives are important tools for understanding the performance in magnetic recording systems. Multiple parallel platforms have been used in micromagnetics, including multi-core

central processing units (CPU) and graphics processing units (GPU). In particular, GPUs were introduced offering ultra-high performance, which allowed using inexpensive desktop computers as high-performance computer clusters [16][18]. Important micromagnetic solvers powered by GPU includes FastMag [16], magnum.fe [19], OOMMF [32], MuMax [21], magnum.fd [43], GPMagnet [44], MicroMagnum [45] and etc. Opportunities of accelerating the micromagnetic solvers with GPUs lie in two aspects: specific compute-intensive algorithms and the entire solver. While the most time-consuming micromagnetic solver components can be accelerated by GPU with certain algorithms, there are still benefits offloading all the computational work within a solver to the GPU [21].

As pointed out by previous sections, the performance brought by GPUs for some important algorithms, such as SpMVM, FFT, NUFFT and N-body, is impressive compared with the same algorithm on CPUs. These algorithms are heavily used in micromagnetic solvers. Taking the magnetostatic field evaluation in the FastMag as an example, the SpMVM algorithm can be applied to the gradient and divergent operator to evaluate the magnetic charge density from the magnetization, and evaluate the magnetic field from the magnetic field from the magnetic potential, respectively. Meanwhile, the NUFFT algorithm is essential to guarantee high speed here by calculating the magnetic potential from the magnetic charge density. The formulation, implementation details and the numerical results for the above algorithms will be addressed in Chapter 3.

Provided with the highly optimized algorithms, having a full GPU version of micromagnetic solver could push the speed performance further. The time spent on the repetitive pointwise array operations and GPU-CPU memory transfers could be saved by

offloading the entire solver onto GPU. Moreover, the basic time integration solvers and some preconditioners are also good for GPU acceleration, as discussed in Chapter 4. An example of a full-GPU micromagnetic solver is covered in Chapter 5.

Chapter 2, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, W. Cui, M. Hu, R. Chang, M.J. Donahue, V. Lomakin, "*Finite Difference Micromagnetic Solvers with Object Oriented Micromagnetic framework (OOMMF) on Graphics Processing Units*," IEEE Transactions on Magnetics, vol. 52, no. 4, pp. 1-9, 2016. S. Fu, R. Chang, S. Couture, M. Menarini, M.A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay, V. Lomakin, "*Micromagnetics on High-performance Workstation and Mobile Computational Platforms*". Journal of Applied Physics, vol. 117, no. 17, pp. 17E517, 2015.

3. Fast Algorithms for Micromagnetic Field Evaluation

Among all the governing interactions, the evaluation of the magnetostatic and exchange energy/field takes most of the effort to develop fast algorithms. For the magnetostatic interaction, we present two methods in this chapter: NUFFT in the context of the FEM and a scalar method in the context of the FDM. Both algorithms are designed to compute the magnetic potential from the magnetic charge densities. The methods reduce the computational complexity of the integral operator evaluation for the magnetostatic field from $O(N^2)$ to $O(NlogN)$.

Differential operators are also essential in micromagnetic solvers, since the application includes the divergence and gradient operators in the magnetostatic field evaluation, and also the exchange field evaluation. In this chapter, we introduce a Sparse Matrix-Vector Multiplication (SpMVM) method on GPUs. In the context of implicit time stepping, SpMVM is called multiple times in each simulation time-step when solving stiff problems. Therefore, the speed of the exchange field evaluation is important for the speed performance of micromagnetic solvers. The capacity of GPU memory has been a limiting factor for the fast algorithms on GPU for a long time. Therefore, the SpMVM algorithm is further developed to run on multiple GPUs. Moreover, a memory-saving approach for SpMVM algorithm on single GPU is introduced. The numerical results of the developed algorithms are addressed to prove the efficiency.

3.1. Fast Magnetostatic Field Evaluation

The magnetostatic field can be defined as a superposition integral

$$\mathbf{H}_{\mathrm{ms}}(r) = \nabla\nabla \cdot \iiint_{V'} \frac{\mathbf{m}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} dV' . \tag{3.1}$$

It is one of the most important components of any micromagnetic solver. It can affect both the numerical accuracy and computational speed. Since the magnetostatic interaction involves all-to-all superpositions, the computational complexity of the micromagnetic modeling for it is $O(N^2)$ if the superpositions are evaluated directly. For FEM codes, typically, this integral is replaced by the equivalent Laplace equation solver with a proper boundary handling [46]. Researchers have also developed other approaches to accelerate the evaluation of such interactions, such as the fast multipole methods [47], the non-uniform grid interpolation method [48], and the Fast Fourier Transform [49]. Among them, the Fast Fourier Transform (FFT) which reduces the complexity to $O(NlogN)$ is of great importance and widely adopted by the micromagnetic community in FDM solvers. This is also due to the easy availability of the well-developed numerical FFT libraries [50][51][52].

In this work, the magnetostatic field is implemented by defining magnetic charge densities, magnetic potential and magnetic field via the following procedure:

$$\begin{aligned} &\rho_M = -M_s \nabla \cdot \mathbf{m};\ \sigma_M = M_s \hat{\mathbf{n}} \cdot \mathbf{m} \\ &\phi(\mathbf{r}) = \iiint_V \frac{\rho_M(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} dV + \oiint_S \frac{\sigma_M(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} dS' . \\ &\mathbf{H}_{\mathrm{ms}}(\mathbf{r}) = -\nabla\phi(\mathbf{r}) \end{aligned} \tag{3.2}$$

Here, ρ_M and σ_M are volume and surface charge densities, respectively. The numerical operators that are involved here are differential operators (divergence and gradient operators) and integral operator. In FastMag, the differential operators are implemented as SpMVM, while the integral operator is implemented via the NUFFT algorithm.

To give a good coverage of the fast evaluation methods for the magnetostatic field, a demonstration of the method in the context of the FDM solver will be discussed. The method introduced also follows the three steps in Eq. (3.2). Compared with the traditional method, it saves GPU memory and number of operations. The overall performance of the novel method and the comparison to the traditional method are addressed. To achieve the best speed performance, all the algorithms in this chapter are implemented on CPU and GPUs.

3.1.1. NUFFT for Finite Element Method

The integral operator in Eq. (3.1) is a convolution between the magnetization $\mathbf{m}(\mathbf{r}')$ and the Green's function $1/|\mathbf{r}-\mathbf{r}'|$. The brute-force way to tackle the evaluation would be of the complexity of $O(N^2)$. Fast Fourier Transform can reduce the computational complexity of the convolution from $O(N^2)$ to $O(NlogN)$ and it is easy to implement using well-developed FFT libraries such as FFTW, Intel MKL and Nvidia CUFFT library available online.

However, the regular FFT algorithms available from the above packages are insufficient for the non-uniform distributed source problems. The non-uniform distributed

sources are common in the solvers with FEM, when the magnetization is defined at the finite element mesh nodes (vertices of the tetrahedrons).

An approach to reduce this high computational cost is NUFFT, which extends the FFT from uniform to general non-uniform sampling. Apart from the regular FFT algorithm, NUFFT introduces the interpolation and projection procedures to handle the non-uniformly distributed sources accompanied with near-field corrections. Here, we use "sources" and "observers" to represents the input and output of the NUFFT algorithm, respectively.

To be specific, NUFFT solves the non-uniformed distributed sources problem by subdividing the computational domain into uniform boxes, so that FFT can be operated after projecting/interpolating the sources to vertexes of boxes. NUFFT takes advantage of the fact that the magnetostatic far-field varies slowly. As a result, the magnetostatic far-fields can be approximated well even if we do not compute the far-field with the exact locations of sources. On the other hand, the boxes within a pre-defined distance for a certain observer point are considered as near-field boxes and the field generated by the sources inside near-field boxes are computed directly with analytical method.

The NUFFT algorithm comprises four stages. In stage I, the sources are interpolated/projected onto the vertexes of uniform boxes. FFT is, then operated on the sources to get the potential on vertexes of boxes in stage II. In the next stage, the field/potential is interpolated/projected to the observer points. In the last stage, the near-field is computed analytically and added to the previous results.

To implement NUFFT on GPU, we utilized Nvidia CUFFT library for the FFT computation. Other kernels are optimized such that the GPU memory is accessed in a

coalesced way, which means that a group of GPU global memory can be read/write in parallel. Furthermore, the fast-speed GPU shared memory is also utilized to reduce the slow-speed GPU global memory accesses within, for example, the forward and backward interpolation/projection kernels. These optimization methods are important because memory access is the bottleneck for most of GPU kernels that we have implemented for NUFFT.

To realize the algorithm on GPU efficiently, the algorithm has to be carefully implemented as summarized in the following steps:

A. Projection.

Due to the limitation of the FFT algorithm, any irregularly located sources have to be projected onto regular grids first. Therefore, a uniform grid is created to hold the projected values from the sources. The grid comprises the number of grid points of *O(N).* Defined by the grid points are boxes. Each box contains a certain number of non-uniform sources (or observers) such that this number is of $O(1)$.

As shown in Fig. 3.1, a non-uniform source point (green triangle) is projected onto the surrounding uniform grid points. Note that the uniform source projection grids have to enclose all the sources in space. In Fig. 3.1, the range of non-uniform distributed sources (green solid line) is within the regular source grids (black solid line). The interpolation method uses Lagrange polynomials [53], among which the linear interpolation (first order interpolation) in 2D space is shown as an example in Fig. 3.1. Two interpolation schemes, namely linear interpolation and cubic interpolation (third order interpolation), are implemented in our NUFFT algorithm. There are speed and accuracy differences out of the choice of the interpolation order. Due to the amount of

numerical operations required for the computations, it is clear that the cubic interpolations are more accurate while the linear interpolations are faster. A practical choice of the interpolation method is based on the application. For instance, in the context of the most applications we run with FastMag, linear interpolation is accurate enough. Therefore, linear interpolation is preferred in such cases and we set linear interpolation as our default option for NUFFT algorithm.

Although the CPU implementation of this procedure seems straightforward, having an efficient GPU implementation is not simple. Coalesced memory access should be followed for reading the amplitudes and coordinates of the sources. Shared memory is used to hold the data, which helps to only access data from slow global memory once. All the repetitive memory access is through the shared memory. We note that all the necessary data are pre-loaded from CPU to GPU global memory. This also applies to the rest of the implementation steps.

The computational cost for all boxes and sources scales as *O(N)*.

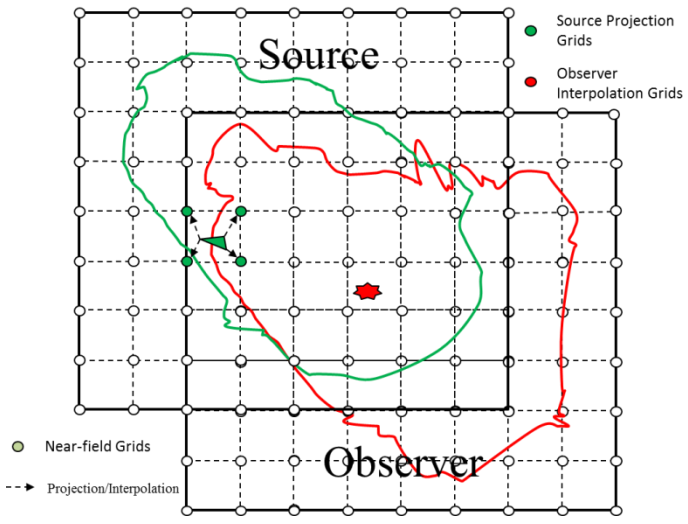

Figure 3.1: Projection step of NUFFT. The randomly distributed source (green triangle) is projected to the uniform surrounding grid points.

B. Fast Fourier Transform.

The next step is to apply the FFT algorithm on the regular grids. In Fig. 3.2, the convolution between the sources (green circles) and the Green's function $1/|\mathbf{r}-\mathbf{r}'|$ is evaluated and results in the potential at the observer points, which are located on the uniform observer grid (red circles).

Nvidia cuFFT library [52] is utilized for computing FFTs on GPUs. In FFTs, we exploit the fact that the real-space sources and potentials are real-valued quantities to minimize the GPU memory consumption and computational workload. The CPU implementation uses the FFTW library. The interface of FFTW is similar to that of cuFFT and it provides a good performance on CPU benchmarks [50].

Since only FFTs are involved in this stage, its computational cost is of *O(NlogN)*.

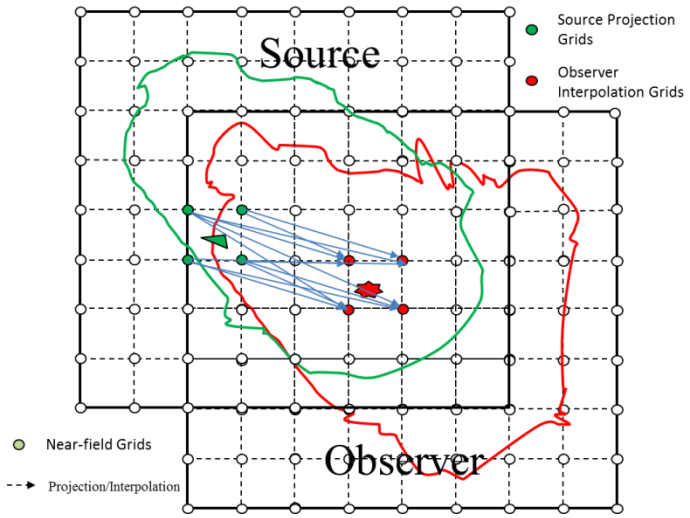

Figure 3.2: FFT step of NUFFT. Demonstrate FFT to compute the convolution from source grids to observer grids.

C. Back-projection.

This is an inverse process to step A, applied to the observers. Lagrange polynomials are used to interpolate from the uniform observer grids (red circles) to the non-uniformly distributed observer points (red stars). Note that only the surrounding regular grids contribute to the observers inside the same box. The back-projection procedure provides linear and cubic interpolation options to provide the freedom to balance between the speed and numerical accuracy.

Similar to the projection stage, utilizing the shared memory and following the coalesced memory accessing rule is the key to implementing an efficient GPU kernel for this stage. The computational cost of this stage is of *O(N)*.

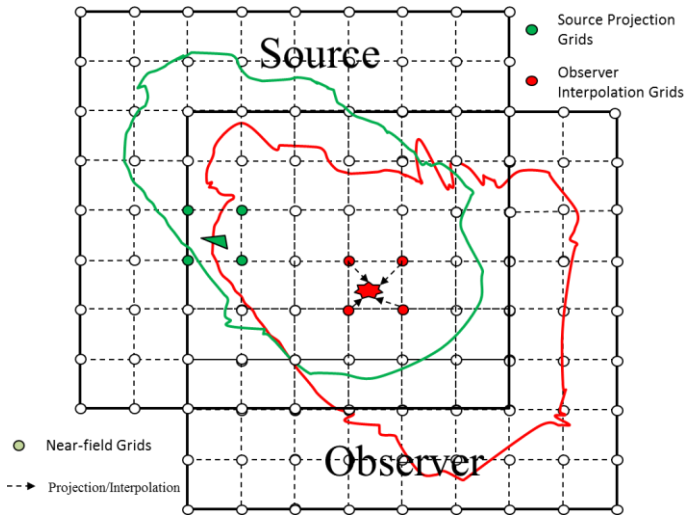

Figure 3.3: Back projection step of NUFFT. Interpolate the FFT results from previous step (red circles) to the non-uniformly distributed observers (red star).

D. Near-field correction.

The last near-field correction step is necessary to ensure the accuracy of the NUFFT algorithm. By applying the FFT on the regular grids, location shift from the original sources and observers to the uniform grids causes numerical errors when sources and observers are close to each other. To solve this problem, the contributions of the

potentials from the nearby sources are re-computed in the near-field correction step. To be specific, the FFT results are subtracted from the results of the previous step and then analytically computed superposition integrals from the near-field sources are added. Within the range of the near field correction, the complexity of this step is $O(N_{nf}^2)$, where N_{nf}^2 is the number of sources (and observers) per box. However, we choose the number of boxes as $O(N)$ so that the number of observers per box is $N_{nf} = O(1)$. As a result, the overall computational effort of this stage for all boxes is of $O(N)$.

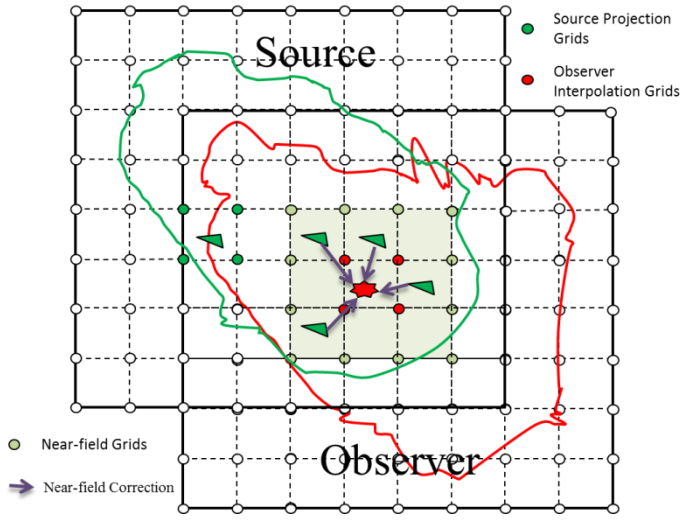

Figure 3.4: Near-field correction step of NUFFT. Subtract the FFT results from the nearby grids and then add analytical results back. The accuracy of the NUFFT method is guaranteed by this step and is tunable by defining the range of the nearby boxes (the range of the light green boxes).

Through defining the range of the nearby grids, the time spent in this step and the accuracy of the NUFFT algorithm are controllable (the accuracy control is in addition to the accuracy control by the interpolation order). In practice, we take the surrounding boxes around a certain observer box as near-boxes, as indicated by the light green shaded range in Fig. 3.4. The range of these boxes allows controlling the error. In many cases

one level of boxes is sufficient for accuracy at the level of 1e-3 and two or three levels may be needed for accuracies levels of 1e-4 – 1e-5. There is also a compromise if it is more efficient to control the accuracy by increasing the interpolation order or by increasing the near-box range.

3.1.2. A Scalar Potential Approach for Finite Difference Method

This section introduces a scalar potential approach for computing the magnetostatic field in the context of FDMs. This approach is an alternative to a more conventionally used "tensor" approach, which directly finds the magnetostatic field using superposition with the tensor integral kernel. The scalar potential approach uses scalar charges to find the scalar potential, which is used to compute the field as the gradient of the potential. The scalar potential approach has a lower computational cost and memory requirements. In this section, we compare their formulation, GPU implementation, numerical accuracy and speed performance.

A. Tensor Method

In the tensor formulation, the double del operator is moved under the integral from (3.1) to result in

$$\mathbf{H}_{\mathrm{ms}}(\mathbf{r}) = M_s \iiint_{V'} \left(\nabla\nabla \cdot \frac{1}{|\mathbf{r}-\mathbf{r}'|} \right) \cdot \hat{\mathbf{m}}(\mathbf{r}') dV' . \tag{3.3}$$

For numerical implementation, the magnetization is assumed to be uniform in each discretization cell and the field is obtained by averaging over the observation cells:

$$\mathbf{H}_{\mathrm{ms},i} = \sum_{j=1}^{N} \mathbf{N}_{ij} \mathbf{m}_j ; \quad \mathbf{N}_{ij} = V_i^{-1} \int_{S_i} \int_{S_j} \frac{d\mathbf{S}_i d\mathbf{S}_j}{|\mathbf{r}_i - \mathbf{r}_j|} , \tag{3.4}$$

where V_i is the volume of the of the cell i , whereas $d\mathbf{S}_i$ and $d\mathbf{S}_j$ represent surface integrals over the surfaces of the cells i and j . The tensor $\mathbf{N}_{ij}$ provides the field generated by the magnetization in the cell j at the cell i and can be computed as outlined in [54].

The direct cost of computing the magnetostatic field via Eq. (3.4) is of $O(N^2)$. However, using the fact that the discretization is uniform, the summation in Eq. (3.4) can be computed via three-dimensional Fast Fourier Transform (FFT), which reduces the computational cost to $O(N \log N)$.

The numerical procedure involves one forward FFT of the $\mathbf{N}_{ij}$ tensor components, which is done once in the precomputation step. In each time integration step, the computations involve forward FFTs of the three vector **m** components, products and summations of these components with the Fourier images of the tensor $\mathbf{N}_{ij}$, and the inverse FFT of the three vector components of the resulting magnetic field.

The number of operations of the tensor method per time step scales as $3c_1 N \log N$, where the constant c_1 is related a single scalar 3D FFT evaluation. The memory storage scales as $12N$. The rate of numerical convergence for decreasing cell size D is of $O(D^{-2})$ [55].

B. Scalar Potential Method

In the scalar potential approach, the field is evaluated by finding the volume and surface charges, computing the potential, and finding the field via finite differences as Eq. (3.2). The potential is found via the scalar superposition integrals.

Evaluating the volume charges is a straightforward task. On the other hand, evaluating the surface charges and the corresponding scalar potential is more involved. This is because if the magnetization locations are assigned at the cell centers, the surface charge locations are shifted by half-cell with respect to the volume charges. One simple approach, then, is to consider volume and surface charges and corresponding potential parts separately. However, such an approach would require evaluating the integrals for the potential twice. The approach we use is to find the magnetization states at the grid nodes by averaging over cells surrounding the nodes, i.e.

$$\mathbf{m}_{i_n} = \sum\nolimits_{i_c} \mathbf{m}_{i_c(i_n)} \Big/ I_c(i_n), \tag{3.5}$$

where for each node i_n the summation is over the cells $i_c(i_n)$ surrounding this node and $I_c(i_n)$ is the number of the surrounding cells. These nodal magnetization values are then used to find the surface charges at the grid nodes at the boundaries. Next, surface charges are found as a sum of the surface charges on the surfaces surrounding the nodes:

$$q_{s,i_n} = \sum\nolimits_{i_s(i_n)} (\mathbf{m}_{i_n} \cdot \hat{\mathbf{n}}_{i_s(i_n)}) s_{i_s(i_n)}, \tag{3.6}$$

where i_n is the boundary node numbers, $\hat{\mathbf{n}}_{i_s(i_n)}$ is the normal corresponding to the surrounding boundary surfaces, and $s_{i_s(i_n)}$ is the areas of the surrounding surfaces. The areas $s_{i_s(i_n)}$ are taken such that no part of any surface is account more than once per each surface node, i.e. if the structure is discretized into cubic cells with the side surface area of Δs, $s_{i_s(i_n)} = \Delta s/4$. The volumetric charges q_{v,i_n} corresponding to the nodes are found by finite differences at all nodes. Through similar approach as surface charges, there is no overlap between volumes taken for different q_{v,i_n}. In this approach the volumetric and

surface charges are collocated at the nodes, which allows lumping them together for the purpose of computation of the scalar potential:

$$\phi_{i_n} = \sum_{\substack{j_n=1 \\ \mathbf{r}_{i_n} \neq \mathbf{r}_{j_n}}}^{N_n} \frac{q_{j_n}}{|\mathbf{r}_{i_n} - \mathbf{r}_{j_n}|} + \phi_{i_n}^{self} , \tag{3.7}$$

where $q_{j_n} = q_{v,j_n}$ at the interior nodes and $q_{j_n} = q_{v,j_n} + q_{s,j_n}$ at the boundary nodes. The term $\phi_{i_n}^{self}$ represents the self-term given via the exact integration of the volume and surface charges corresponding to the effect of the surrounding surfaces and volumes on the same node. The computation can be further made more accurate by considering that the charges q_{j_n} are spread into volumetric charge densities in cells centered at the node and replacing the factor $|\mathbf{r}_{i_n} - \mathbf{r}_{j_n}|^{-1}$ with the integral $\int_{V_{j_n}} (q_{j_n} / \Delta v_{j_n}) |\mathbf{r}_{i_n} - \mathbf{r}_{j_n}|^{-1} dv$. The summation in Eq. (3.7) or the integrals in Eq. (3.2) can be treated as three dimension scalar convolutions via FFT. Once the potential at the nodes is found, the magnetostatic field is found by finite differences at the centers of the cells.

As compared to the tensor approach, the scalar potential approach has a reduced computational cost and memory consumption. In particular, the number of FFTs is reduced from 6 to 2, and the memory consumption is reduced 3 times. The overall accuracy of computing the field scales quadratically with respect to the discretization cell size, similar to the tensor approach, although as shown in the examples the tensor approach is more accurate by a constant for the same cell size.

C. Implementation

Provided that the GPU memory is limited compared to the CPU memory, saving memory is important to enable large-scale problems. This is especially important in the

implementation of the tensor approach for the magnetostatic field. The GPU memory is mainly consumed by storing the Green's function and magnetization or magnetic charge data in Fourier-transformed space. For *N* discretization cells, the storage required for the tensor approach is *30N* real numbers, including *24N* for the 3 FFT-extended vector components of the magnetization and *6N* for the FFT-extended tensor; this storage calculation includes zero padding for the non-cyclic convolution, symmetries, and the fact that the computation space is real. The storage of the scalar potential approach in [56][57][58] is *27N* real numbers, including *24N* for the 3 FFT-extended vector components of the magnetization and *3N* for the FFT-extended Green's function. The memory requirement for the scalar potential approach presented here is significantly lower—it is *12N* real numbers, including *8N* for the FFT-extended scalar potential, *3N* for the magnetization, and *1N* for the FFT-extended scalar Green's function. The GPU memory cost of both the tensor approach and scalar potential approach can be further reduced by 1/3 or more with the FFT approach introduced in [59]. Such an improvement would maintain the fact that the scalar potential approach is more favorable in terms of computational speed and memory consumption. The GPU memory consumption is carefully managed by reusing GPU memory whenever possible, such that extra GPU buffer is rarely needed. Up to 8M cells and 4M cells can be fit into a 2GB GPU with the scalar potential method and the tensor method, respectively.

D. Numerical Results

(1) Accuracy Analysis

Fig. 3.5 compares the accuracy of the magnetostatic field evaluation via the scalar potential and tensor approaches. In order to validate the accuracy and convergence, we

discretize a fixed-size cube (49nm x 49nm x 49 nm) into an increasing number of cubic cells. The initial magnetization state $\mathbf{M} = z\hat{\mathbf{z}}$ was chosen so that both volume and surface charge densities exist (ρ_M = -1 emu/cm4, σ_M = 49 emu/cm3 when z = 49 nm, else σ_M = 0). For this case, the magnetostatic field can be found analytically. The error was defined as

$$Error = \sqrt{\frac{\sum_i^N \left|\mathbf{H}_{\mathrm{ms,i}}^{num} - \mathbf{H}_{\mathrm{ms,i}}^{ana}\right|^2}{\sum_i^N \left|\mathbf{H}_{\mathrm{ms,i}}^{ana}\right|^2}}. \tag{3.8}$$

From Fig. 3.5 it is evident that both the scalar potential and tensor approaches have quadratic convergence. The tensor method is more accurate due to the fact that it avoids the approximation of charges at the boundaries and numerical derivative operations in the superposition integrals.

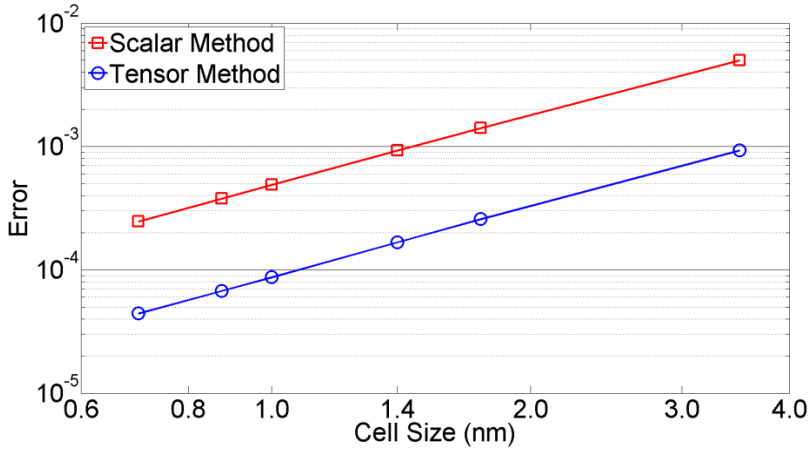

Figure 3.5: The numerical error of GPU implementation for the magnetostatic field by the scalar potential method and tensor method as a function of discretized grid cell size. Both methods show a quadratic convergence.

(2) Speed Comparison

We can clearly compare the speed of scalar and tensor methods implementations of the solver in Fig. 3.6. The running time per time step of the implementations of both methods follows a $O(N \log N)$ trend. We can also observe that the speed of the scalar method on GPUs is higher than that of the tensor method. At the points where CUFFT has the highest efficiency on GPU, e.g. 8^3, 16^3, 32^3, 64^3, 128^3, the scalar method is about 50% faster. This percentage is higher at other points (up to ~60%) because CUFFT is working with lower efficiency at these points and that leads to a higher weight of FFT comparing to other parts of the code. As a result, the problem of more FFTs becomes more significant at these points. Since FFT performs better when the transformed array size are composed of small prime factors, like 2, 3, 5 and 7, we always zero-pad the array to these sizes. As a result, a smooth curve of timing results is achieved in Fig. 3.6.

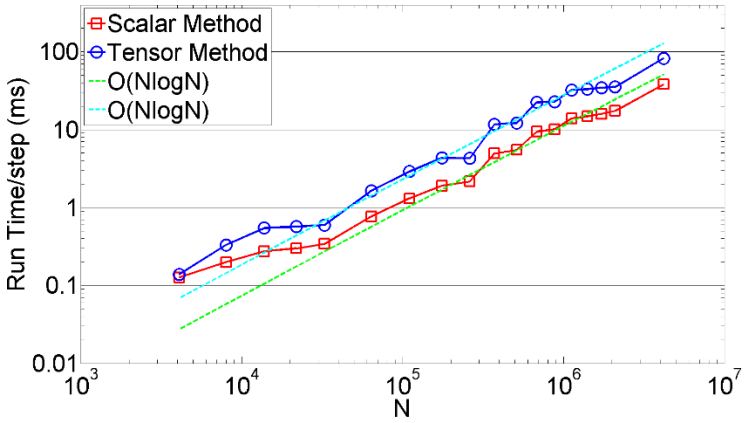

Figure 3.6: Simulation time per time step for the scalar potential method and tensor method as a function of problem size. The scalar method is faster than tensor method and both results fit well with the *O(NlogN)* trend.

3.1.3. GPU OOMMF with Tensor Approach

As an extension of the work on the GPU implementation of the tensor method, we demonstrate a GPU implementation of the widely used Object Oriented Micromagnetic Framework (OOMMF) based on the tensor method [20]. The implementation is such that most of the user-related OOMMF components are unchanged and only the lower-level modules are ported to GPU. This allows OOMMF users to run their models as before but at a greater speed. The GPU-accelerated OOMMF has been made freely available to the micromagnetic community at the UCSD website [61] and OOMMF website [60].

A. Implementation

In addition to the implementation details discussed with respect to the tensor method above, here, we discuss details related to the time evolver. The time evolver is the section of code that implements the time evolution of the LLG equation. To avoid CPU-GPU data transfers at every time step, we implemented the time evolver on the GPU so that the entire OOMMF simulation runs on the GPU. The adaptive Euler method and a fixed-time-step evolver were implemented. The adaptive time evolver includes error-tracking kernels. The reduction kernel, which sums up and finds the minimal or maximal values of an array, is required for numerical error-tracking in the adaptive time evolver. It is not an easy kernel to implement efficiently on the GPU because it requires significant data communication between CUDA threads and it is not compute-intensive. A highly efficient GPU reduction implementation [62] was adopted. With this reduction kernel, the global memory is read via coalesced access to shared memory. The shared memory is then used for the reduction with serial addressing to avoid shared memory bank conflicts. In addition, synchronization among CUDA threads is avoided to the extent possible by

taking advantage of the "atomic" behavior of the GPU warps. This approach results in a highly efficient reduction kernel as demonstrated in the following section.

B. Numerical Results

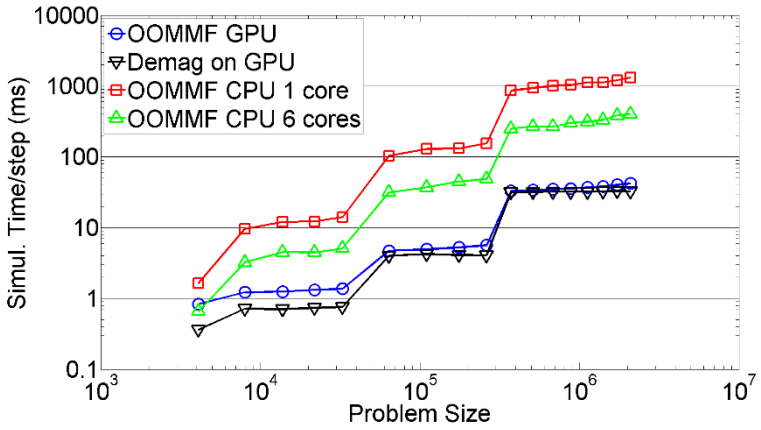

Figure 3.7: Runtime per time step for OOMMF on CPU and GPU as a function of the number of discretization cells N. The time for the magnetostatic field computation on the GPU is also included. The computation for the magnetostatic field takes most of the run time in the GPU implementation.

Fig. 3.7 shows the timing results of the OOMMF adaptive Euler solver using single-precision for the GPU computations and double-precision for the CPU computations. There is a difference in the simulation time of OOMMF running on CPU versus GPU, but both show a step-like behavior in the simulation time. The steps occur when the number of cells in each dimension surpasses a power of two, i.e. 16, 32, 64. This occurs because OOMMF pads the FFT array to a power of two. For example, when the number of grid cells is 33, the FFT array is padded to 128 although a size of only $2\times 33-1=65$ is necessary for the computation. With this padding strategy, the FFT computation always stays at its best performance, whereas there are some unnecessary computations during the simulation.

Fig. 3.7 also breaks down the time spent on the magnetostatic field computation on GPU. This time is very close to the computational time of the entire OOMMF solver on GPU when the total number of discretized cells is large enough. This reflects the fact that in our implementation kernels other than the magnetostatic field are subdominant. One can also observe that the computational time for the magnetostatic field has higher weight at the points with sizes that are not powers of two. This further verifies that the FFT computations take most of the computational time when the FFT array is padded to a power of two.

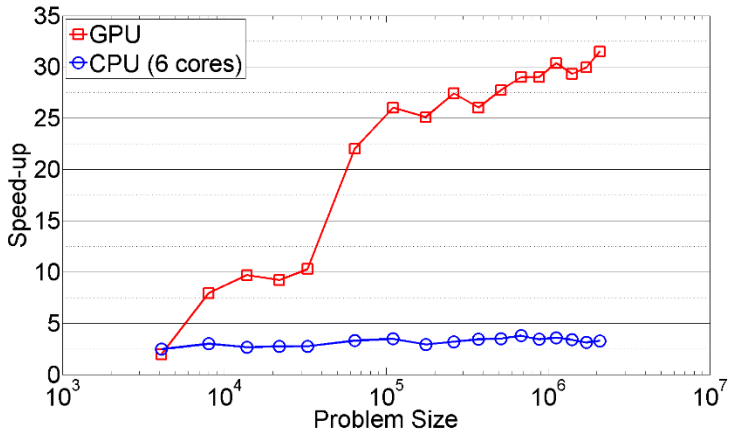

Figure 3.8: GPU and multi-core CPU speed-up of OOMMF implementation as a function of the number of discretization cells N. An increase in the speed-up with N is observed.

Fig. 3.8 shows the GPU-CPU speed-up, demonstrating the speed-up increase with the number of discretized cells. The efficiency increase is due to the fact that multiple GPU streaming processors can be utilized more efficiently for larger problems and the memory access time is hidden by the computations to a larger extent. In the same figure, limitations of speed-ups by multi-core CPU is observed.

We also tested double-precision computations as shown in Table 3.1. We find that using the GTX690 the double-precision performance is 2.0x – 3.5x slower than single-precision. It is interesting to note that the number of double-precision streaming processors on the GTX690 GPU we used is 24x fewer than single-precision processors. The comparatively smaller reduction of the double-precision performance indicates that the FFT computations that dominate the overall cost are memory access latency limited. Indeed, the memory access time for a given number of double-precision accesses is about twice that for the same number of single-precision accesses. The reduction in the computational speed for the double-precision case is closer to 1/2 as explained by memory bandwidth and not 1/24 as would be explained by the number of streaming processors.

Table 3.1: Timing Results of OOMMF Solver

N	OOMMF					
	CPU 1 core (ms)	CPU 6 cores (ms)	GPU Single prec (ms)	Speed-up	GPU Double prec (ms)	Speed-up
4K	1.63	0.66	0.84	2.0	1.61	1.3
32K	14.11	5.12	1.37	10.3	2.69	5.2
256K	155.3	48.71	5.67	27.4	16.62	9.4
2M	1323	401.8	41.96	31.5	136.9	9.7

3.2. Fast Exchange Field Evaluation

The Laplacian operator involved in the evaluation of the exchange field can be extracted as a Sparse Matrix Vector Multiplication (SpMV) process. In the context of implicit time stepping, SpMVM is called multiple times in each simulation time-step when solving stiff problems. Therefore, the exchange field evaluation efficiency is

important for the speed performance of a micromagnetic solver. The formulation of SpMVM we are going to discuss in this section is

$$y = \mathbf{A}x \tag{3.9}$$

where **A** is a sparse matrix, **x** is a dense input vector and **y** is a dense output vector. A sparse matrix means that the number of nonzero entries is much fewer than the total number of entries in the matrix. In order to save memory and reduce computation complexity, only nonzero entries are stored and computed in the SpMVM method. However, due to indirect and irregular memory access pattern resulting in bad spatial locality [63], careful implementation strategies have to be taken to yield high SpMVM performance on GPU.

Though having a strong computation power, GPU is limited by its memory size to solve ultra-large problems. GPU memory capacity is relatively small comparing to the size of CPU RAM that can be installed on a node. The scarcity of GPU memory becomes serious in micromagnetic solvers where there are several GPU algorithms consuming the memory. For example, in FastMag, GPU memory is shared by dense matrix algorithms like NUFFT and sparse matrix algorithms. To solve this problem, we developed a memory saving approach, which is named “on-the-fly” method because it transfers the matrix part by part on-the-fly. As it is time-consuming to transfer memory from host to device, we take advantage of the concurrent CUDA streams strategy provided by Nvidia toolkit to overlap the memory transfer time with computational time. In addition, CUDA pinned-host-memory is exploited to maximize the memory transfer throughput. Within a single GPU, our memory saving approach enables solving ultra-large SpMVM problems with rather high performance.

Implementing SpMVM on multiple GPUs is another solution to the GPU memory size limitation. Multi-GPU parallelism can further accelerate the simulation speed. Through sorting the sparse matrix before computations, we divide the input vector evenly among GPUs so that the memory scalability can be preserved. Workload balance among GPUs was carefully accounted for so that high parallel efficiency is achieved.

3.2.1. Acceleration Strategy

A. Storage Format

The High-Performance Computer (HPC) community has been exploring several methodologies to implement SpMVM on GPUs, especially on the storage formats related topics. Bell and Garland have reported benchmarks of the performance of SpMVM on GPU with a variety of storage formats, such as Coordinate (COO), Compressed Sparse Row (CSR), Blocked-CSR (BCSR), Diagonal (DIA) and ELL formats [64]. Among them, CSR format simply compresses the nonzeros in the row order and gives a steady good performance on various sparse matrices. Considering that we are focusing on the effectiveness of our multi-GPU implementation and memory saving approach on single GPU, a straightforward implementation of CSR becomes our first choice.

CSR format stores the value and column index of each nonzero element in arrays *Data* and *Ptr*, but the nonzero elements are stored row-wise so that the row index of each nonzero element does not need to be explicitly kept in memory. Instead, a shorter array *RowOffset* stores the index of the first nonzero entry in each row, and the last element of *RowOffset* is the total number of nonzeros elements in the matrix. For a $M \times N$ matrix with the total number of nonzero entries being *NNZ*, the length of *Data* and *Ptr* is *NNZ*

while the length of *RowOffset* are *M+1*. Fig. 3.9 illustrates the zero-indexed CSR storage format of an example matrix. The arrays *RowOffset* and *Ptr* work as lookup tables so that the position of each nonzero entry can be identified.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 9 \\ 0 & 8 & 0 & 0 \\ 0 & 2 & 7 & 0 \\ 0 & 0 & 3 & 5 \end{bmatrix}$$

$$\text{RowOffset} = \begin{bmatrix} 0 & 2 & 3 & 5 & 7 \end{bmatrix}$$
$$\text{Ptr} = \begin{bmatrix} 0 & 3 & 1 & 1 & 2 & 2 & 3 \end{bmatrix}$$
$$\text{Data} = \begin{bmatrix} 1 & 9 & 8 & 2 & 7 & 3 & 5 \end{bmatrix}$$

Figure 3.9: An example of the sparse matrix CSR format.

B. Sorting Sparse Matrix

Reordering the rows and columns of the sparse matrix is an effective way to improve the performance of both single GPU and multi-GPU implementations. First, the sparse matrix sparsity pattern can be improved via grouping the nonzero entries together. A better data uniformity leads to coalesced memory access of input vector **x**, which is important because the memory access speed is the bottleneck of SpMVM algorithm on GPU. Second, conducting the reordering of sparse matrix helps keep the memory scalability of multi-GPU and multi-stream implementations. Taking the *M*-GPU implementation as an example, ideally, only $1/M$ of input vector should be sent to each GPU. However, in reality there are always overlaps between input vector pieces on difference GPUs due to the uncertainty of nonzenonzeroro element position in the sparse matrix. Sorting helps reduce the overlap between input vector pieces by grouping the nonzero elements of the sparse matrix along the diagonal, thus increasing the certainty of nonzero element position. Third, sorting the sparse matrix can help with the workload balance for multi-GPU and memory saving implementations. This is because the sparsity

pattern on different chunks of sparse matrix is guaranteed to be similar by reordering the matrix.

There is a broad range of reordering algorithms proposed such as Column Count Method [65], Approximate Minimum Degree Method (AMD) [66], reverse Cuthill-McKee Method (RCM) [67], King's Algorithm [68], and the Traveling Salesman Problem (TSP) [69]. Among them, AMD method and RCM method are only for symmetric matrices. We proposed a sorting method based on the fact that the sparse matrices and vectors in the considered applications are generated from sources with definite coordinate information in three-dimensional space and the fact that nonzero elements are generated by the interactions among adjacent sources. For example, the exchange field in the micromagnetic solvers is represented by the Laplacian operator on the magnetization. The Laplacian operator can be translated into sparse matrix while the magnetization becomes the dense input vector. In this case, the nonzero elements represent the exchange interaction among local magnetizations. Taking these into account, we can develop a sorting algorithm that groups the nonzero entries together around the diagonal in the matrix, without the constraint of symmetry of sparse matrices.

Since the nonzero entries in the sparse matrix come from the adjacent sources, we reorder the sources by cubic boxes so that all the sources located in the same box lead to a successive alignment of their corresponding nonzero elements in the sparse matrix and input vector. To be more specific, first, we define the boundaries of the sources in Cartesian coordinates system. Then, we divide them into uniform sized boxes, which are denoted with continuous numbers to specify the box order. The sources in the first box are aligned as the first group of elements of the input vector, and then the sources in the

second box and so on are aligned. Since the sources are sorted by box, we name this sorting method as “box-sorting method”.

To illustrate the effectiveness of the box-sorting method, we present spy-plots of a sparse matrix generated for exchange field in the Finite Element Method, as shown in Fig. 3.10. The blue dots in the figure represent the non-zero elements in the matrix. The apparent difference between the width of nonzero elements bands in Fig. 3.10a and Fig. 3.10b shows the effectiveness of the box-sorting method. Since we are testing on a symmetric matrix, we also show the sorting result by RCM in Fig. 3.10c. We need to clarify that our box-sorting approach is not as elegant as other approaches like RCM, but it coincides well with the sorting method used in FastMag solver. Provided that the GPU speed is almost the same with our box-sorting approach and RCM, we adopted our simple but effective sorting approach in this paper.

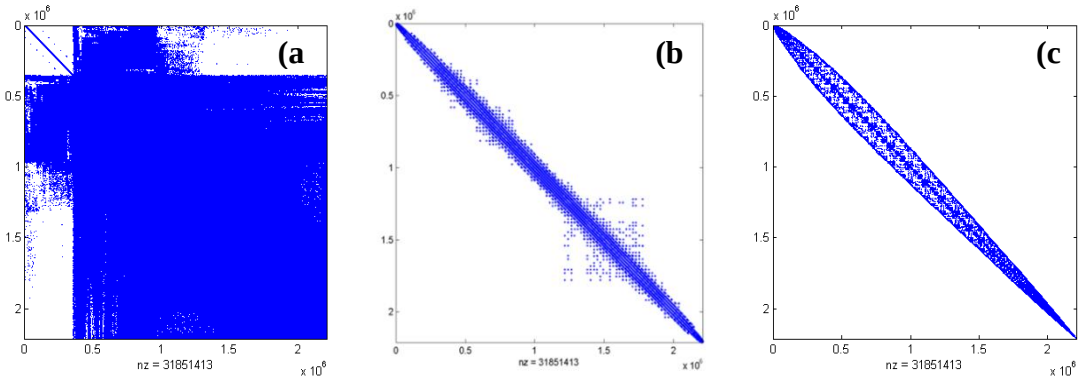

Figure 3.10: Spy-plots of one sparse matrice before and after sorting. (a) spy-plot of unsorted sparse matrix (b) spy-plot of the sorted sparse matrix with box-sorting method (c) spy-plot of the sorted sparse matrix with RCM sorting method.

C. Single GPU Memory Saving Approach

Since the GPU memory capacity is relatively small comparing with CPU RAM, the problem size that can be addressed by GPUs can be significantly limited if all the

matrix elements are kept in the GPU memory. There are two ways to solve the problem. The first way is to use multiple GPUs. The second approach is to use a single GPU but transfer memory and compute piece by piece. However, the performance of the latter method can be limited by the speed of the CPU-GPU memory transfer, which is for a PCI-E 3.0 is typically around 4x to 5x slower than the memory speed of a CPU [70]. In order to ease the limitation of the GPU memory size while maintaining a high speed, we developed an on-the-fly method running on a single GPU. The basic concept is to transfer part of *Data* and *Ptr* at one time to GPU instead of transferring the entire sparse matrix to GPU at the preprocessing step. One decides on the number of pieces that the *Data* or *Ptr* are divided into, each of which is assigned to a separate CUDA *stream*. Here, a *stream* is a sequence of operations that execute in issue-order on the GPU, and operations in different streams can run simultaneously on a GPU. We use the term "flying stream" to denote how many streams are allowed to run simultaneously/on-the-fly. Fig. 3.11 takes an example composed of 8 streams with 2 flying streams to illustrate how the on-the-fly method works. At the beginning, only two streams are assigned to transfer the memory and execute GPU kernels. Once *stream0* is done with computation, *stream2* is ready to take over the memory space from *stream0* for memory transfer and computation. Therefore, half of the GPU memory space is passed among streams with even numbers, while the other half is passed among streams with odd numbers. As shown in Fig. 3.11, the capability of overlapping computation time and memory transfer time is utilized to mitigate the time loss by memory transfer.

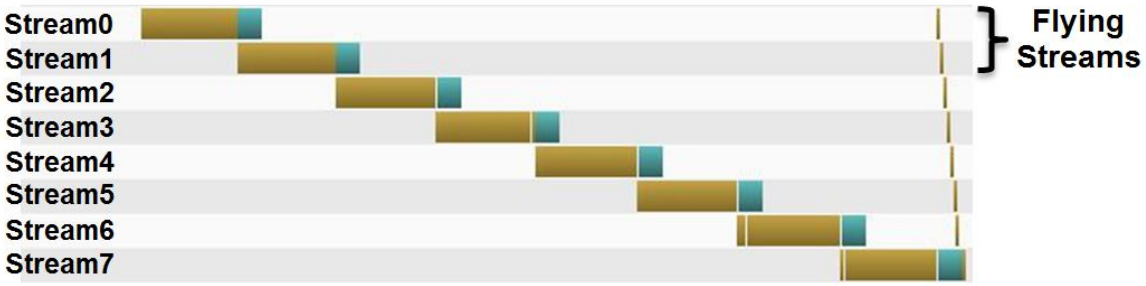

Figure 3.11: Run-time streaming of an 8 streams with 2 flying streams implementation shown in Nvidia Visual Profiler. Yellow strips represent the memory transfer and green strips represent kernel computation.

The total number of streams N_{stm} equals the number of workload division, while the number of flying streams that is denoted as N_{fly_stm}. If we use M to denote the total amount of memory needed without on-the-fly method, then about M/N_{stm} is needed by each stream. In order to overlap the memory transfer and GPU kernel time, the least possible value for N_{fly_stm} is 2. Thus we only need the GPU memory size of $N_{fly\_stm} \times M / N_{stm}$ to conduct the computation. In other words, if we define S as how many times the device memory is saved by on-the-fly method, then

$$S = \frac{M}{N_{fly\_stm} \times M / N_{stm}} = \frac{N_{stm}}{N_{fly\_stm}} \tag{3.10}$$

However, the memory saving is not free. The total amount of transferred memory increases. As a result, the speed of memory saving approach is slower than regular single GPU implementation. On the other hand, as we can find from Fig. 3.10 there are overlaps between input vectors on different streams. With the increase of N_{stm}, the total amount of overlap will also increase. In other words, Eq. (3.10) is a good estimate of the memory-saving rate. To reduce the influence from input vector overlap, the assignment of sub-matrices and input vectors to streams is also based on the sorting method.

Different from single GPU and multi-GPU implementations where only *InputVector* needs to be transferred iteratively, memory saving approach requires all GPU data arrays being transferred on-the-fly. Data is transferred and computed on GPU piece by piece and operations for different pieces could be executed in parallel through CUDA streams. Moreover, CUDA streams allow overlapping of operations on different streams. To be more specific, the computational time is overlapped by memory transfer time in our implementation, as shown in Fig. 3.11.

Apart from CUDA stream, CUDA event is another essential component of memory saving approach. Assuming we have two flying streams and $flyID = streamID \%2$, *cudaStreamWaitEvent()* can guarantee that there will not be race conditions among streams with the same *flyID*. The details of our implementation are shown by the pseudo-code listed in Pseudocode 3.1. The code can be divided into three parts: the first part is data uploading from host to device, the second is kernel launching and the last part is data download from device to the host. Memory download is put into a separate loop to avoid interference with memory upload. CUDA compiler analyzes the dependence of the issues declared in each stream to decide all the possible overlapping between computation and memory transfer. As illustrated by Fig. 3.11, the resultant overlap between kernel and memory transfer is favorable.

The price of transferring memory on-the-fly is the increase of total amount of memory being transferred. As shown in Fig. 3.11, the speed of memory saving approach is mainly decided by memory transfer speed. With that said, utilizing pinned-host-memory to maximize the memory throughput is essential to improve the performance of memory saving approach. Pinned-host-memory is always ready to be fetched by GPU

without the help of CPU, which is called Direct Memory Access (DMA). On the contrary, normal host memory transfer to GPUs has to be interfered by CPU. According to our test, pinned-host-memory transfer throughput can be 12.5 GB/s which essentially doubles the throughput of normal host memory. Apart from memory throughput, no CPU interference decides that memory transfer of pinned-host-memory can be asynchronous to CPU operations. As a result, GPU kernel can be overlapped with memory transfer, as we discussed in the last paragraph.

```
for streamID = 1:numStreams
      flyID = streamID % 2
      StreamWaitEvent( Event[flyID] )
      MemcpyHostToDevice( flyID )
      SpMVMKernel<<< gridSize, blockSize, sMemSize, streamID >>>()
      RecordEvent( Event[flyID] )
end for

for streamID = 1:numStreams
      MemcpyDeviceToHost( streamID )
end for
```

Pseudocode 3.1: Memory saving approach.

3.2.2. Implementation of the computation

A. SpMVM Kernel

Since multi-GPU and memory saving approach reuse the same GPU kernel, an efficient single GPU kernel is the prerequisite of the other two implementations to yield high performance. In this section, we introduce the implementation details of our SpMVM GPU kernel.

There are two different methods to implement CSR kernel computation. One is referred as scalar CSR, the other is called vector CSR [71]. Scalar CSR uses the strategy

of “one thread per row” while the vector CSR is called “multi-threads per row” approach. It has been proved by Bell and Garland that the vector CSR method is more efficient in most cases [71] and according to our test the vector CSR method is over 3x faster than scalar CSR method. Therefore, we adopt the vector CSR method in our demonstration of SpMVM.

The number of threads per row in the vector CSR method is possible to be 2, 4, 8, 16 or 32, according to the matrix sparsity pattern. The denser the matrix, the more threads are assigned to each row. The selection of the number of threads per row is important to guarantee most CUDA threads to be active during computation. The maximum number of threads per row is 32 because the smallest parallelization unit in GPU is a warp which is composed of 32 threads. We limit the number of the warp to be 1 to synchronize the instructions in a block. If there are more than 32 nonzero elements in a row, the operations will be processed serially.

SpMVM kernel can be roughly divided into two steps: dot product and reduction. Dot product step is the multiplication of nonzero elements in the sparse matrix and corresponding input vector elements. The involvement of accessing *Data*, *Ptr*, *RowOffest* and *InputVector* arrays decides the bottleneck be memory access speed. With vector CSR format, memory access of *Data* and *Ptr* arrays are coalesced, while access of *RowOffset* array is an efficient broadcast among threads. However, the access of *InputVector* is non-coalesced because of the unpredictable position of the corresponding sparse matrix element. Sorting before computation could improve the memory access efficiency by allowing threads in a warp to access adjacent elements of *InputVector*. Provided that our sorting method has effectively reduced the band of the sparse matrix, there is a good

cache locality in the access of *InputVector*. However, L1 cache is not a choice to cache global memory in the Nvidia Pascal architecture. Considering that the access of *InputVector* is read-only, texture memory is helpful to enforce GPU to cache *InputVector* so that the memory traffic between SM and L2 cache is reduced.

The second step of SpMVM kernel, which is the reduction of dot product results in each matrix row, is implemented in a parallel way presented by M. Harris [62]. This strategy takes advantage of all possible threads to do a reduction, instead of limiting the entire workload to only one thread which is used in the scalar method. This parallel reduction strategy relies on communicating intermediate reduction results between threads. As we have introduced, shared memory or shuffle instructions are the available two options for the communication. In our implementation, we select shuffle instruction in order to get rid of the synchronization between threads and the memory traffic between registers and shared memory. Another reason why we want to avoid the shared memory is to keep the possibility of modifying the kernel in future without the worry about limited shared memory resource in SM.

B. Multiple GPUs

Now that we have developed an efficient SpMVM kernel, the next step to achieve a successful multi-GPU implementation is to take care of the sparse matrix partition. For those large sparse matrix problems that cannot fit in a single GPU, multiple GPUs can handle them by dividing the sparse matrix and input vectors into separate chunks and solve them separately. Provided that we are using the CSR format, it was proven in [72] that partition the matrix by row is superior to partition by column or grids for multiple

GPU implementations. Therefore, we followed this strategy so that each GPU can work on one chunk of sparse matrix independently. The parallelization among GPUs is accomplished via OpenMP so that each CPU thread controls one GPU. In this way, the operations on different GPUs can be launched at the same time.

Workload balance is an important issue for high parallel efficiency among GPUs. Here, the workload not only means the number of nonzero elements but also the sparsity pattern. In our implementation, the number of nonzero elements is leveraged among chunks of the matrix. However, the sparsity pattern is hard to be perfectly the same among chunks. Here, sorting the sparse matrix can guarantee that the nonzero elements are grouped along matrix diagonal in each chunk so that the sparsity pattern is more likely to be the same than the original sparse matrix.

3.2.3. Numerical Results

A. Single GPU Results

To demonstrate the performance of our SpMVM kernel, we listed our regular single GPU implementation testing results in Table 3.1. The necessity to put the computation on GPU was proven by the fact that our single GPU implementation is up to 14.0x, averagely 12.1x faster than MKL. In addition, based on the fact that our single GPU implementation is up to 1.7x and averagely 1.6x faster than *cusparse* library, we can say that we have reached a high parallel efficiency in our GPU implementation. Comparing to *cusparse* library, our GPU kernel alone is 1.2x faster and the memory transfer is 1.9x faster.

Texture memory efficiently improved our kernel performance by caching the *InputVector* in the texture cache. Taking the test case of FEM Sphere as an example, texture cache reduces the memory traffic from L2 cache to SM by 2.1x from 2.1GB to 974MB. The similar story happens to other test cases. Thus it has been proved that binding the irregular memory access of *InputVector* array to texture memory is beneficial.

The sparsity pattern has a big impact on the sparse matrices whose nonzero elements spread randomly, like the case of FEM Sphere. The spy-plots of a sorted and unsorted matrix of this test case were shown in Fig. 3.10, where we can observe a significant difference. Comparing with 5.54 ms that is needed for the sorted case, unsorted sparse matrix takes 16.51 ms. Better sparsity pattern improves the L2 cache hit rate by 3.1x from 19% to 59%, whereas memory traffic from global memory to L2 cache is reduced from 1180MB to 718MB. The texture memory hit rate is also increased by 3.3x from 19% to 62%. The memory access to texture memory is more coalesced, such that the memory traffic from the texture memory to SM is reduced by 22% from 836MB to 635MB. Since high cache hit rate and coalesced memory access reduces the memory access time, there are more eligible warps available for warp scheduler in SMs to parallelize the workload. Therefore, the parallel efficiency is significantly improved.

Table 3.2: Computational time of single and multi-GPU implementations.

	nnz/ (nnz/row)	SpMVM				Serial MKL	Parallel MKL	GPU cuspar se
		1 GPU	2 GPU	3 GPU	4 GPU			
FEM Cube	17.5M/26.5	2.72	1.63	1.22	1.07	38.02	4.79	4.53
FEM Sphere	31.8M/14.4	5.54	3.53	2.49	1.81	74.72	11.99	8.99
dielFilterV3 real	89.3M/81.0	9.57	5.18	3.59	3.37	123.1	18.93	13.42
gsm_10685 7	21.8M/37	3.38	1.99	1.54	1.20	39.95	4.96	5.16
cube_coup_ dt6	124M/58	16.5	8.89	6.46	4.89	139.0	31.40	25.40

The application of pinned host memory also guarantees the speed of our implementation. To execute the same amount of computational operations, sparser matrices need to transfer more data as compared to the computation. In other words, memory transfer plays a more important role in such cases. According to our test, the pinned host memory leads to up to 12.49 GB/s CPU-GPU memory throughput, while the non-pinned host memory only performs up to 4.53 GB/s memory throughput. Therefore, the pinned host memory method is necessary for SpMVM in solvers that have heavy data transfer workload.

B. Multiple GPU results

The speed scaling of the multi-GPU implementation can be observed from Table 3.2. The multi-GPU code shows a continuous speed-up with the increase of the number of GPUs. For example, the performance is 1.81 ms with 4 GPUs solving FEM Sphere problem, corresponding to 6.6x faster than the execution of MKL running with 12 CPU threads.

Based on the definition of parallel efficiency

$$E_p = S_p / P = T_1 / PT_p \tag{3.11}$$

where E_p is parallel efficiency, S_p is speed-up, P is the number of GPUs, T_1 and T_p are the execution time of sequential and parallel algorithms respectively, we can observe a high parallelization efficiency across multiple GPUs with our implementation. To be specific, the parallel efficiency is up to 84.4% and averagely 73% using 4 GPUs in five test cases.

The parallel efficiency decreases with the increase of the number of GPUs as shown in Table 3.2 because (a) the workload cannot be kept perfectly balanced among all GPUs and (b) CPU-GPU memory transfer bandwidth is shared by multiple GPUs. Although we kept the number of nonzero elements balanced among GPUs and sorting method greatly improves the sparsity pattern of the entire sparse matrix, the pattern is impossible to be exactly the same among sparse matrix chunks. As a result, kernel workload is not the same among GPUs. On the other hand, memory transfer speed is not improved much since memory bandwidth is shared, while the computation time is significantly reduced. Therefore, memory transfer plays a more important role in the performance. With the help of sparse matrix sorting, the sizes of subsets of *InputVector* that needs to be sent to GPUs are kept scalable with the number of GPUs such that multi-GPU parallel efficiency is improved.

C. Single GPU Memory Saving Approach Results

From Table 3.3 we can find that the device memory consumption continuously decreases with the increase of the number of streams. Due to the fact that some arrays cannot be only allocated for flying streams, the actual device memory consumption is

further away from the ideal scaling with the increase of the number of streams. However, the decrease of the device memory consumption is still high, if we define the memory saving efficiency as

$$E_m = \frac{M_I}{M_c} = \frac{M/S}{M_c} = \frac{M}{M_c} \cdot \frac{N_{fly\_stm}}{N_{stm}}, \tag{3.12}$$

where M_I is the ideal memory consumption calculation based on Eq. (3.10), M_c is the actual memory consumption, M, N_{fly_stm} and N_{stm} are all from Eq. (3.10). Apparently, when N_{stm} equals N_{fly_stm} the memory is not saved so it can be taken as the baseline for comparison. With 32 streams and 2 flying streams, the memory saving efficiency is up to 88.8% and averagely 79.0%. The efficiency is higher in the case *dielFilterV3real* because the nonzero elements are all efficiently grouped around the diagonal of the matrix after sorting so that the memory consumption is balanced among all the streams.

From the testing results in Table 3.2 and Table 3.3, memory saving approach is up to 2.8x, averagely 2.3x faster than CPU MKL. Provided that MKL is a highly optimized library designed for Intel CPUs, we are confident that our memory saving approach is efficiently implemented. We can also find that the memory saving approach has a steady speed when applying different numbers of threads. However, memory saving is not always free, the computational time starts to increase substantially when the number of threads is greater than 32.

Table 3.3: Speed and device memory consumption of memory saving approach.

Num. of Streams		2	4	8	16	32
FEM Cube	Speed/ms	15.20	14.67	14.47	14.52	14.82
	Memory/MB	148.23	76.20	39.58	21.26	12.08
	E_m^*	100.00%	97.26%	93.63%	87.15%	76.7%
FEM Sphere	Speed/ms	28.10	27.14	26.93	27.59	29.03
	Memory/MB	287.522	150.06	80.84	46.00	28.60
	E_m^*	100.00%	95.80%	88.92%	78.13%	62.8%
dielFilterV3real	Speed/ms	63.76	61.75	60.86	61.31	60.39
	Memory/MB	729.16	367.79	187.42	97.13	51.32
	E_m^*	100.00%	99.13%	97.26%	93.84%	88.8%
gsm_106857	Speed/ms	16.54	15.87	15.26	15.65	15.62
	Memory/MB	184.74	94.81	48.82	26.61	14.67
	E_m^*	100.00%	97.43%	94.60%	86.78%	78.7%
cube_coup_dt6	Speed/ms	97.29	94.89	93.68	94.62	95.42
	Memory/MB	1041.17	528.40	269.01	139.17	74.23
	E_m^*	100.00%	98.52%	96.76%	93.52%	87.7%

D. SpMVM in FastMag

We used FastMag on the real-world test cases to test the effect of the SpMVM algorithms introduced above. Fig. 3.12 shows a very basic construction of magnetic write head. The tiny tip (~nm) of magnetic write head (~um) in hard drive has to be defined by sharp tetrahedrons to achieve good accuracy, where the numerical stiffness is generated. In such kind of micromagnetic problems, implicit time evolving method requires calling sparse matrix multiplication algorithm 3-20 times.

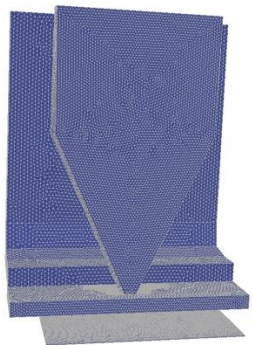

Figure 3.12: Schematic view of a magnetic write head.

We simulated a writing process in the hard drive, the goal of which is to observe the dynamics of magnetization on the write head. A total of 0.58 million nodes and 2.4 million tetrahedron elements are generated to mesh the magnetic write head. The simulation takes 572 time-steps to run from 0 ps to 210 ps, where more than 11 thousand times of sparse matrix multiplications are involved. (We note that this mesh was relatively stiff and the number of linear iterations can be reduced by preconditioning.) The testing results are listed in Table 3.4. When we compare the results of SpMVM on single GPU with MKL on CPU, we obtain 10.1x of acceleration in the sparse matrix multiplication, which leads to 1.6x acceleration of entire FastMag solver. The MKL on CPU for SpMVM takes 33.4% of the total solver running time, which is the bottleneck of solver speed. With GPU version of SpMVM, it only takes 5.3% of the total running time. In other words, running SpMVM on GPU solves one of the bottleneck of FastMag solver in this case.

A similar simulation with larger problem size, where regular single GPU implementation does not fit into GPU memory, is tested with multi-GPU implementation and memory saving approach. The mesh of the magnetic write head contains a total of 3.3 million nodes and 18.1 million tetrahedron elements. The simulation takes 697 time-steps to run from 0 ps to 210 ps, where more than 30 thousand times of sparse matrix multiplication are involved. (We note that this mesh was relatively stiff and the number of linear iterations can be reduced by preconditioning.) The FastMag solver achieves a 1.2x speed-up through launching 4 GPUs, comparing with MKL on CPU with 12 CPU threads. The speed-up is caused by 6.4x acceleration in the sparse matrix multiplication section.

When we use 32 streams for memory saving approach to saving the GPU memory by ~16x, a 1.3x acceleration was achieved for entire FastMag solver, comparing with MKL on single CPU thread. The speed-up is led by a 2.3x acceleration in the sparse matrix multiplication section. Therefore, both multi-GPU implementation and memory saving approach on single GPU are proven to be capable of solving the ultra-large problems, while achieving high computational performance.

Table 3.4: Computation Time of FastMag Solver (in seconds).

		Regular Single GPU	Multiple GPUs	Memory Saving Approach	Serial MKL	Parallel MKL
Write Head Small	Solver	158.2	N/A	N/A	254.9	N/A
	SpMVM	8.4	N/A	N/A	85.3	N/A
Write Head Large	Solver	N/A	5868.2	7817.5	10483.1	6821.6
	SpMVM	N/A	189.4	1718.0	4014.0	1209.7

In summary, we introduced SpMVM algorithms accelerated by GPU for ultra-large MM simulations. Based on an efficient single-GPU kernel, we developed multi-GPU implementation and memory saving approach to overcome the limitation of GPU memory size. Through embedding our GPU SpMVM in the micromagnetic solver, we accelerate the solver by 1.6x. Solving large MM problems are enabled with multi-GPU implementation and memory saving approach on a single GPU, with a high performance.

Several interesting points were addressed:

(1) We take advantage of the CSR format as the data structure of the sparse matrix. Corresponding to the data structure chosen, we assign several threads to each row of matrix instead of one thread per row.

(2) Sorting is important for SpMVM in many aspects: a) speed up kernel by coalesced memory reading and the improvement of the cache hit rate b) balance the workload for multi-GPU implementation c) high memory saving efficiency with memory saving approach.

(3) Texture memory can release the heavy non-coalesced memory access burden on the L2 cache and it shows better performance while binding InputVector to texture memory.

(4) Page-locked memory not only leads to a higher memory copy throughput but also allows the possibility of overlapping memory transfer time with kernel computational time.

(5) CUDA streams allow us to transfer memory from host to device on-the-fly, which is the basis of our memory saving approach. CUDA event provides the possibility to control the issue order between streams, which is also essential for our implementation.

We demonstrated considerable performance gains over other high-performance CPU and GPU libraries. With a single GPU, we achieved 14.0x faster than single-threaded MKL implementation and 1.7x faster than Nvidia cusparse library by single GPU. With 4 GPUs, as much as 6.6x and 5.2x are demonstrated comparing with multi-threaded MKL and cusparse library, respectively.

The memory saving approach saves the GPU memory by up to 14.2x, corresponding to 88.8% memory saving efficiency. The speed of memory saving approach is averagely 2.3x faster than MKL on single CPU thread.

The memory saving approach we proposed is not limited to CSR format, and it fits a large variety of sparse matrix formats. In the meantime, our implementation of

SpMVM is not limited to micromagnetic applications, other scientific areas such as electromagnetism, molecular dynamics and fluid dynamics can easily take advantage of our SpMVM algorithm to solve ultra-large problems.

Chapter 3, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, W. Cui, M. Hu, R. Chang, M.J. Donahue, V. Lomakin, "*Finite Difference Micromagnetic Solvers with Object Oriented Micromagnetic framework (OOMMF) on Graphics Processing Units*," IEEE Transactions on Magnetics, vol. 52, no. 4, pp. 1-9, 2016.

4. Fast Algorithms for Time Integration In Micromagnetic Solvers

While the field/energy evaluation has been extensively studied by the researchers [53][56][57], the time integration is another key aspect of a micromagnetic solver. We take the solver component that takes charge of the time integration as time evolver. A time evolver not only affects the speed performance but also determines the robustness of evaluation of the magnetic dynamics. Given the same type of field/energy evaluation methods and implementation, an efficient time evolver may deliver a much better performance to stay at the same accuracy level. On the other hand, a bad time evolver can easily diverge in complicated test-cases.

This chapter starts with an introduction to the basics of time integration, including explicit and implicit time integration methods, as well as linear multi-step methods and Runge-Kutta methods. Then, specific methods in the context of the implicit time integration method used in FastMag are discussed, namely, the backward differential formula (BDF), Newton iteration method, Generalized Minimal Residual (GMRES) method equipped with system Jacobian evaluation. The last but not the least, an efficient preconditioner designed for stiff problems in micromagnetic simulations is introduced. The efficiency of the preconditioner is investigated with several practical test-cases.

4.1. Explicit and Implicit Time Integration Methods

LLG equation evolves in time by integrating a set of Ordinary Differential Equations (ODE). A general form of ODE can be written as

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}, t), \tag{4.1}$$

where $\mathbf{y}$ is a column vector of unknowns, f represents a set of functions and t is time. For the LLG equation, the unknown $\mathbf{y}$ in the ODE Eq. (4.1) is the magnetization vector.

Provided with an initial value of magnetization vector, the LLG equation evolves in time and reach a certain state. The methods to evolve in time can be categorized into two classes: explicit time integration methods and implicit time integration methods. Explicit time integration methods, such as the Adams-Bashforth method, were first proposed and studied [73][74][75][76]. The explicit methods use the information from current and previous step(s) to update the magnetization for the next step. For example, in the formulation of the simplest Runge-Kutta methods, the Euler method, the derivative in Eq. (4.1) is approximated by a forward difference

$$\frac{d\mathbf{y}_n}{dt} = \frac{\mathbf{y}_{n+1} - \mathbf{y}_n}{dt}. \tag{4.2}$$

Combining Eq. (4.1) and Eq. (4.2), the unknown vector at the next time step $\mathbf{y}_{n+1}$ can be derived from the current vector $\mathbf{y}_n$, right-hand side of Eq. (4.1) $f(\mathbf{y}_n, t_n)$ and the time step size dt :

$$\mathbf{y}_{n+1} = \mathbf{y}_n + dt \cdot f(\mathbf{y}_n, t_n). \tag{4.3}$$

It is obvious that the computational cost in such time evolver, which involves pointwise vector addition and multiplication, is low. In other words, the explicit methods come with the benefits of easy formulation and fast computational speed per time step. However, the time step may become intolerably small in stiff systems using explicit methods [77][78]. Here, an ODE is stiff if the solution being sought varies slowly while there are approximate solutions that vary fast or diverge. The stiffness is related to the differences in the speed of the fastest decay and slowest decay eigen modes [24].

Implicit methods are essential in these scenarios to attain large time steps while preserving the numerical stability The backward Euler method is a simple example for the formulation of the implicit method. The derivative in Eq. (4.1) is approximated by backward differences

$$\frac{d\mathbf{y}_{n+1}}{dt} = \frac{\mathbf{y}_{n+1} - \mathbf{y}_n}{dt}. \tag{4.4}$$

Combining Eq. (4.1) and Eq. (4.4), the numerical evolver solution becomes

$$\mathbf{y}_{n+1} = \mathbf{y}_n + dt \cdot f(\mathbf{y}_{n+1}, t_{n+1}). \tag{4.5}$$

In Eq. (4.5), the unknown vector $\mathbf{y}_{n+1}$ appears on both sides of the equation. Therefore, the solution for a set of nonlinear equations is required. Eq. (4.5) can be linearized with Newton method in the context of Backward Differential Formula (BDF), which will be covered in the following sections. In general, the implicit methods provide better numerical stabilities and much greater time steps. On the other hand, the implicit methods are more complicated in terms of the formulation and implementation, hence the computational cost per step is higher than that of explicit methods. For stiff systems,

however, implicit time integration schemes typically are much more efficient, especially for FEM solvers.

4.2. Linear multi-step methods and Runge-Kutta Methods

The time integration methods can be categorized into explicit methods and implicit methods but also in another way: linear multi-step methods and Runge-Kutta methods. Concretely, there are explicit and implicit multi-step methods and similarly explicit and implicit Runge-Kutta methods.

Runge-Kutta methods use the information from one previous step, and take a few intermediate steps to achieving a certain order accuracy. A general form of Runge-Kutta methods can be written as

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\sum_{i=1}^{s} b_i \mathbf{k}_i \\ \mathbf{k}_i &= f\ (t_n + c_i h,\ y_n + h\sum_{j=1}^{N} a_{s,j}\mathbf{k}_j) \end{aligned}, \tag{4.6}$$

where h is the step size at the current step, $c_i h$ decides the location of the intermediate steps, $\{a_i\}$, $\{b_i\}$ and $\{c_i\}$ make the differences between various Runge-Kutta methods with the same order, the number of intermediate steps s decides the order. Here a method is of order s when its local truncation error at each step is of order $O(h^{s+1})$ while the total accumulated error is or order $O(h^s)$. Note that $N = i - 1$ formulates an explicit Runge-Kutta method from Eq. (4.6), while $N = s$ leads to an implicit form of Runge-Kutta methods. Popular Runge-Kutta methods include Euler method [79], Heun method [28], classical Runge-Kutta methods (rk4) [79], and the Runge–Kutta–Fehlberg method [80],

among others. The explicit Runge-Kutta methods can be extended to be adaptive Runge-Kutta methods. The adaptive methods evaluate the local truncation error at each Runge-Kutta step by comparing the results from two methods with higher and lower order. As the result, the step size h can be adaptive to the estimation results of the local truncation error. These methods are also widely used in micromagnetic solvers [20].

The multi-step methods use a linear combination of the information from several previous steps y_i and $f(t_i, y_i)$ to calculate y and the desired current step. A general formulation of the linear multi-step methods can be written as

$$\mathbf{y}_{n+s} + \sum_{i=0}^{s-1} a_i \mathbf{y}_{n+i} = h \sum_{i=0}^{N} b_i f\left(t_i, \mathbf{y}_{n+i}\right). \tag{4.7}$$

Eq. (4.7) gives an explicit formulation when $N = s-1$, while it gives implicit formulation when $N = s$. Choices of s and $\{a_i\}$ $\{b_i\}$ decide the order of the methods and the specific linear multi-step method type, respectively.

Instead of taking a few intermediate steps, the multi-step methods utilize the computed results from the previous steps, which gives the advantages of reducing the computational cost. Three important families of linear multi-step methods are the Adams-Bashforth methods (explicit method), Adams-Moulton methods [81] (implicit method), and Backward Differential Formula (BDF) [28] (implicit method). For stiff problems, the BDF method is found often the most efficient due to its greater stability range for higher orders. It is also noted that implicit multistep methods can be shown to be absolutely stable up to the second order but they are only conditionally stable for orders greater than the second order. Also, typically implicit methods are not used for orders greater than 6 due to a reduced range of their stability for higher orders.

4.3. The Time Integration Methods in FastMag

The choice of time integration methods for a micromagnetic solver is non-trivial and depends on a particular problem. Considering the advantages of saving computational cost and being efficient on stiff problems, the BDF is a good candidate for the micromagnetic simulations. The strong exchange field may result in the ODE stiffness. In the LLG equation, BDF has been reported to be an effective implicit method to increase the time step, such that the speed of the simulation can be greatly improved [77][24]. In the following sections, the BDF used in the FEM based micromagnetic framework FastMag is discussed. The Newton method to solve the non-linear system and the Jacobian-free linear system solver used in the FastMag are also addressed.

BDF methods are implicit linear multi-step methods with $b_{s-1} = b_{s-2} = \ldots = b_0 = 0$ in Eq. (4.7), the other coefficients are chosen so that the method is of order s. Therefore, a general formulation of the BDF can be summarized as

$$\mathbf{y}_{n+s} + \sum_{i=0}^{s-1} a_i \mathbf{y}_{n+i} = h b_s f\left(t_s, \mathbf{y}_{n+s}\right). \tag{4.8}$$

The BDF in micromagnetic simulations is essentially a non-linear system. The Newton iteration method with Jacobian in Krylov subspaces have been reported to be effective in such systems. V. Tsiantos and J. Miles [82] summarized two categories of the Newton methods: 1) Modified Newton methods which explicitly store the coefficient matrix and update the matrix every few Newton iterations; 2) Full Newton methods which compute Jacobian-vector product in every linear iteration, which can be approximated with the Finite Difference scheme to avoid storing the coefficient matrix

explicitly. R. Chang [24] gave an analytical derivation of the system Jacobian with full Newton method, demonstrating that the computation of Jacobian-vector product can be very efficient by one more functional evaluation and several cross products. In the context of full Newton methods, D. Suess et al. [77] reported that an approximation of Jacobian, omitting the magnetostatic field part, can effectively solve the linear system in Krylov subspace.

This section is concerned with solving the LLG equation, which reads

$$\frac{\partial \mathbf{m}}{\partial t} = \frac{-\gamma}{1+\alpha^2}\left[\mathbf{m}\times\mathbf{H}_{eff} + \alpha\mathbf{m}\times\left(\mathbf{m}\times\mathbf{H}_{eff}\right)\right], \tag{4.9}$$

where $\mathbf{m}$ is the normalized magnetization vector, $\mathbf{H}_{eff}$ is the effective magnetic field, M_s is saturation magnetization, γ and α are electron gyromagnetic ratio and damping constant, respectively.

As introduced in the previous section, the implicit method is essential in the simulations with stiffness problem. The method of interest here is the BDF method with a constant leading coefficient [83]. Combine Eq. (4.8) with Eq. (4.9) it can be written as

$$\mathbf{m}_{n+s} + \sum_{i=0}^{s-1} a_i \mathbf{m}_{n+i} = h b_s f\left(t_s, \mathbf{m}_{n+s}\right), \tag{4.10}$$

where s is the BDF order, h and b_s are constants, n denotes the n^{th} time step. Eq. (4.10) can be rewritten in a fully implicit form

$$F_n(\mathbf{m}_n) = \mathbf{m}_n - \mathbf{a}_n - Bf(t_n, \mathbf{m}_n) = 0, \tag{4.11}$$

where $\mathbf{a}_n = \sum_{j=1}^{q} \alpha_j \mathbf{m}_{n-j}$, $B = h\, b_s$. The non-linear system in Eq. (4.11) can be solved with Newton method by successively adding correction vectors to the solution $\mathbf{m}_n$ in the context of the prediction-correction method. To this end, a prediction is made on the initial solution $\mathbf{m}_n^{(0)}$ at the latest time step n. This solution is corrected by Newton iterations via $\mathbf{m}_n^{(m+1)} = \mathbf{m}_n^{(m)} + \mathbf{v}$. The correction vector $\mathbf{v}$ is a solution of the following linear equation

$$\mathbf{A}\mathbf{v} = \mathbf{b}. \tag{4.12}$$

Here, $\mathbf{b} = -F(\mathbf{m}_{n-1})$ is as in Eq. (4.11), whereas $\mathbf{A} = \mathbf{I} - B\mathbf{J}$ is a matrix comprised of the unity matrix $\mathbf{I}$ and the system Jacobian matrix $\mathbf{J} = \partial f(t, \mathbf{m})/\partial \mathbf{m}$. The correction vector $\mathbf{v}$ is in the format of $\left(\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z\right)'$, so the coefficient matrix $\mathbf{A}$ is composed of 3-by-3 sub-matrices corresponding to the interactions among the x, y and z components, respectively.

The linear system of Eq. (4.12) can be solved in Krylov subspaces. The size of Krylov subspaces, which is related to the number of linear iterations, is decided by the spectrum properties of the coefficient matrix $\mathbf{A}$. Due to the stiffness in the complex micromagnetic problems, especially in the highly exchange coupled systems, the ill-conditioned coefficient matrix may lead to a very large number of linear iterations.

4.4. Stiffness Problem and Preconditioning Methods

The speed problem caused by stiffness can be tackled by preconditioning, which turns Eq. (4.12) into an equivalent system

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{v} = \mathbf{P}^{-1}\mathbf{b} \tag{4.13}$$

for left preconditioners or

$$(\mathbf{A}\mathbf{P}^{-1})\,\mathbf{P}\,\mathbf{v} = \mathbf{b} \tag{4.14}$$

for right preconditioners [84]. The preconditioning matrix **P** should be a good approximation to the coefficient matrix **A**.

Incomplete LU (iLU) decomposition method has been reported by different groups to be efficient [19][77][78]. iLU employs incomplete factorization method to obtain an approximation to $\mathbf{A}^{-1}$. Here the factorization is incomplete because certain "fill" elements, which are nonzero elements newly generated during inversion, are ignored. Different strategies on the "fill" elements decide the conditioning quality of the iLU preconditioners. For example, iLU0 [85] drops all the "fill" elements, iLU(p) [86] only keeps the largest p elements after inversion in each row of the matrix, and iLUt method [86] provides an additional threshold by dropping the elements whose absolute value are much smaller than the norm of the corresponding row. iLU is a relatively sophisticated preconditioner and it can effectively approximate $\mathbf{A}^{-1}$ by tuning the parameters. As a result, it could lead to significant reduction of the number of linear iterations. However, each iLU factorization operation may still be slow, especially when **A** is generated in large systems. In addition, it is not known in advance what iLU flavor would or would not work efficiently in a particular situation, which makes iLU hard to use for a typical user.

A diagonal preconditioner (Jacobi method), which only considers the diagonal entries of the coefficient matrices was proposed by V. Tsiantos et al [84]. However, their

numerical experiments failed for two reasons: a) the main cause of the stiffness issue in the system, namely the exchange interaction, was excluded from their preconditioning matrix; b) according to our own experiments, the Jacobi method ignores too much spectrum information from the coefficient matrix. As a result, it fails even if the exchange interactions are included.

In the following section, we present an efficient preconditioner that has a low computational cost per evaluation and factorization and is efficient for a set of practical micromagnetic problems.

4.5. Block-diagonal Preconditioning Method

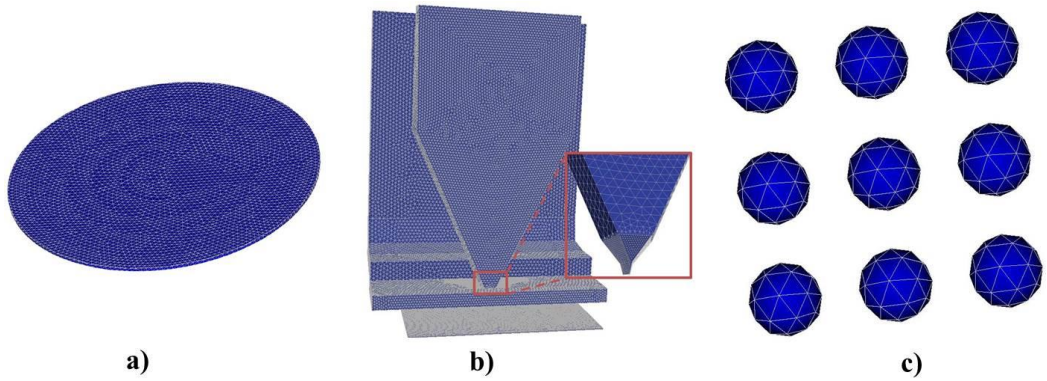

Figure 4.1: Clustered mesh nodes in various applications: a) magnetic thin films, b) magnetic write-head pole tip, c) magnetic particles.

For the applications such as magnetic thin films, magnetic write-head pole tip and magnetic particles, the mesh nodes of the numerical models are “clustered”. For example, each node-pair that resides on the opposite side of the thin films are clustered. The nodes are clustered when they are close to each other while far from the others, as shown in Fig. 4.1 (a). Similar node clusters exist at the pole tip in Fig. 4.1 (b) and each magnetic

particle in Fig. 4.1 (c). The large number of linear iterations reported for similar applications [77] were caused by the strong internal interactions among clustered mesh nodes. Given these facts, we built a light and effective preconditioner, namely block-diagonal preconditioner. We explicitly compute the coefficient matrix, subdivide it into sub-matrices by discarding the close-to-zero elements and then invert each of the sub-matrices. Binding the directly inverted sub-matrices gives an approximation to the inverse of the coefficient matrix. The cost of preconditioning is kept low by 1) always inverting small matrices and 2) being able to update and factorize the preconditioning matrix every few integration steps. On the other hand, the coefficient matrix-vector product exploits the full Newton method in the analytical way to maintain favorable accuracy in the linear system while saving memory consumption. Compared to iLU, block-diagonal method is highly parallelizable since the sub-matrices are disjoint, leading to independent processes such as inversion of the sub-matrices and the sub-matrix vector product. The easy implementation allows highly efficient parallel efficiency on multi-core CPUs or GPUs.

A. Formulation

For applications such as the magnetic thin films, magnetic write-head pole tips and magnetic particles in Fig. 4.1, iLU preconditioner is not optimal since it takes the global interactions into account. In fact, the strong local interactions among clustered mesh nodes are the main causes of the stiffness. Therefore, we developed a fast and easy approach to building an approximation to the inversion of **A** matrix. The method is based on the facts that

1) In the scenario of the stiff problem, the stiffness of the coefficient matrix can be estimated by l_{ex}^2/h_s^2 , where l_{ex} is the exchange length and h_s is the shortest edge of mesh [24]. Thus focusing on the clusters of mesh nodes that locate close to each other should provide enough spectrum information of the coefficient matrix. Fig. 4.2 shows an example where the stiff problem is led by a number of nodes with small h_s . Therefore, it is reasonable to drop the interactions among the weakly-associated nodes. This can greatly save the computational time at the price of minor numerical loss.

2) When A is expressed in the form of blocks, for example $\mathbf{A}=\begin{bmatrix}\mathbf{M} & \mathbf{G}\\ \mathbf{H} & \mathbf{N}\end{bmatrix}$, then $\mathbf{A}^{-1}\approx\begin{bmatrix}\mathbf{M}^{-1} & 0\\ 0 & \mathbf{N}^{-1}\end{bmatrix}$ if $\mathbf{G}\approx 0$ and $\mathbf{H}\approx 0$. Since the computational complexity of inverting a matrix is $O(N^3)$, it is always faster to invert a few small matrices than to invert a large matrix. Thus the capability of inverting the small sub-matrices independently can further accelerate the preconditioner.

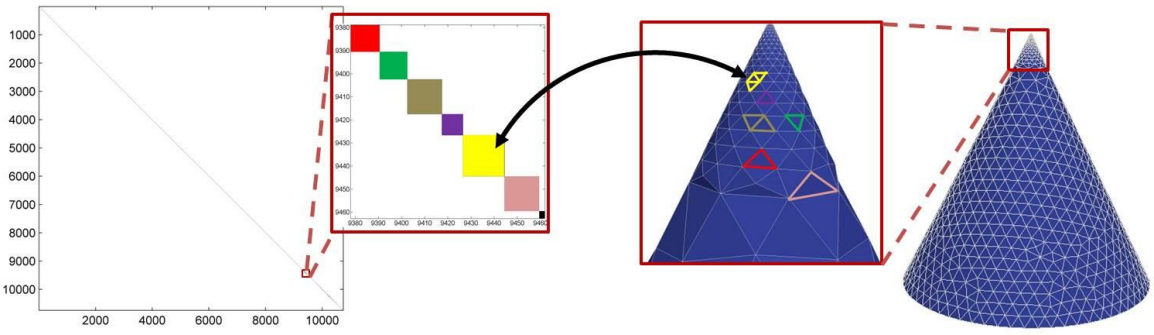

Figure 4.2: Identification of the blocks from the coefficient matrix. The spy-plot (left) of the coefficient matrix extracted from the corresponding tetrahedral mesh (right) is exhibited.

As the granularity of sub-matrices gets smaller, the speed of inversion and multiplication should also get faster. One extreme of the granularity of sub-matrices is that each node is identified as an isolated block to form a single sub-matrix (a 3x3 matrix). We have to mention that this could still give good conditioning results, since any sub-matrices to be inverted contains the necessary information about the appropriate interactions, such as magnetostatic interactions and exchange interactions.

B. Implementation

The implementation of the block-diagonal method can be split into two stages: one-time block-identification operations at the simulation set-up stage; and the coefficient matrix factorization stage during the iterative time-evolving stage.

How a list of "blocks" could be generated is straightforward: the mesh nodes that are geometrically close to each other will be identified as "blocks", as shown in Fig. 4.2. However, the implementation that takes the following steps is not trivial:

1) Define a threshold value. The interactions that are weaker than the threshold will be dropped. A reasonable threshold could be $thresholdValue = thresholdDistance^{-2}$, where *thresholdDistance* could vary based on the mesh information. Though the "interaction" here can include all the interactions involved, it is applicable to only take the exchange interaction, which is the main cause of the stiffness, into account.
2) To formulate a list of blocks, for each node that has not been included in any other blocks,
    a. Add it to a void list, together with its neighbor nodes that have interactions stronger than the threshold value and have not been included in any other

blocks. Here, the interaction strength is represented by the amplitude of the element in the sparse coefficient matrix.

b. Exclude a node from the list above if it has a stronger interaction with another node outside the list. It will be included in another block later on.

c. Add the current list, which is identified as one "block", to a full list composed of blocks.

3) A full list of blocks will be formulated after going over all the nodes. This list will be utilized in the following time integration stage to sub-divide the coefficient matrices into sub-matrices.

During the time integration, the preconditioning matrix is updated when **A** is sufficiently outdated. Updating the preconditioning matrix involves formulating the blocked coefficient matrix based on the list of blocks created earlier and then directly inverse the sub-matrices. The preconditioning could be applied either with the method of Eq. (4.13) or Eq. (4.14). Note that here the matrix **A** to be block-wise inverted is not used as the coefficient matrix in Eq. (4.12).

Compared to the iLU0 method where the memory consumption depends on the number of nonzero element in the coefficient matrix, the storage requirement of both iLUt method and the block-diagonal method scales as $O(N)$, where N is the number of the nodes in the mesh. With a reasonable distance threshold under the block-diagonal method, a preconditioning matrix could be mostly comprised of 3x3 sub-matrices. The corresponding average size of the sub-matrices is usually between 3x3 and 6x6.

4.6. Numerical Results

The efficiency, speed, stability and the scalability of the proposed preconditioner have been examined by a variety of numerical experiments with FastMag.

Magnetic thin films are important structures to be investigated, since they are often used in applications, such as magnetic memories and magnetic recording media. A cone with very non-uniform mesh (50x length ratio between longest and shortest edge) is concerned as a typical example of small stiff problems. We also tested a set of magnetic write heads with identical geometry size, but with varying mesh size and mesh quality. The numerical examples here are designed to cover different problem scales and severity of stiffness issue. For all the simulation shown here, we use a workstation with an Intel Xeon E5 CPU and Nvidia GTX690 GPU. The magnetostatic field evaluation is offloaded onto the GPU, otherwise, all the other numerical computation is conducted on a single core of CPU.

A. Magnetic Thin Films

The vertically aligned mesh-node pairs that reside on the opposite sides of the thin films are the dominant causes of the stiffness problem in FEM simulations. This is because for a very thin film, the opposite nodes are strongly exchange coupled. The blocks identified via the node distances can bind those aligned nodes together, such that the stiffness issues caused by each individual node pair can be tackled accurately. With the block-diagonal method, the strong interactions contained in each node-pair are organized into sub-matrices while other minor interactions are dropped. The sub-matrices can be inverted independently to form a high-quality preconditioning matrix. The choice

to set the threshold distance of the block-diagonal method is to be greater than the thickness but smaller than the lengths of the edges on the surfaces.

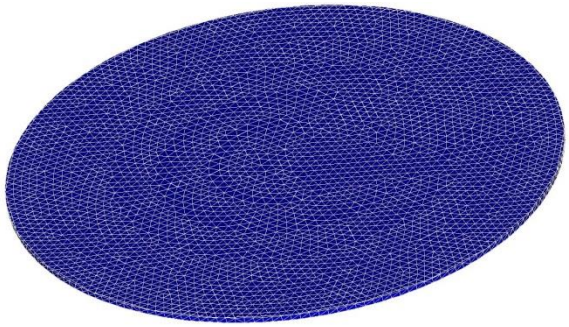

Figure 4.3: Geometry and the mesh of the magnetic thin film test-case.

The relaxation process of a soft thin film is demonstrated in our numerical experiment. The initial out-of-plane magnetization goes in-plane due to the shape anisotropy. The dimensions of the thin film in Fig. 4.3 are 150 nm for the radius and 0.5 nm for the thickness. The average mesh edge length is 5 nm. The magnetic properties include the magnetic exchange constant A_{ex} = 1.0e-6 erg/cm, saturation magnetization M_s = 1068 emu/cc, uniaxial anisotropy $H_k = 2K/M_s = 0$, damping constant $\alpha = 0.5$. The number of the nodes and Finite Element tetrahedral elements in the mesh are 6.7K and 19.4K, respectively. Simulation time is set to be 5 ns. The absolute and relative time step tolerances are both 1e-3, and the linear iteration tolerance is 5e-2.

The stiffness of the coefficient matrix **A** can be represented by the 2-norm condition number $\kappa(\mathbf{A})$ [24]. The condition number $\kappa(\mathbf{A}) = \sigma_{\max}(\mathbf{A})/\sigma_{\min}(\mathbf{A})$ is the ratio of largest singular value to the smallest singular. It depicts the differences between Eigen modes with fastest and slowest decay. Fig. 4.4 clearly shows that the condition number of the coefficient matrix is effectively reduced by the block-diagonal method throughout the

simulation. As a result, the number of linear iterations per Newton step is reduced as well. Fig. 4.4 also reveals the correlation between the condition number of the coefficient matrix to the number of linear iterations involved in the simulation. The out-of-plane magnetization goes in-plane at around 0.1ns, before which the numerical linear system is less stiff since the time-step is relatively small. The time step adaptively increases when the stable final state is reached after 0.1ns. At this stage, the condition number increases dramatically as the numerical system turns stiff. However, the block-diagonal method keeps the condition number at a low level (red curve in Fig. 4.4), regardless of the magnetization dynamics.

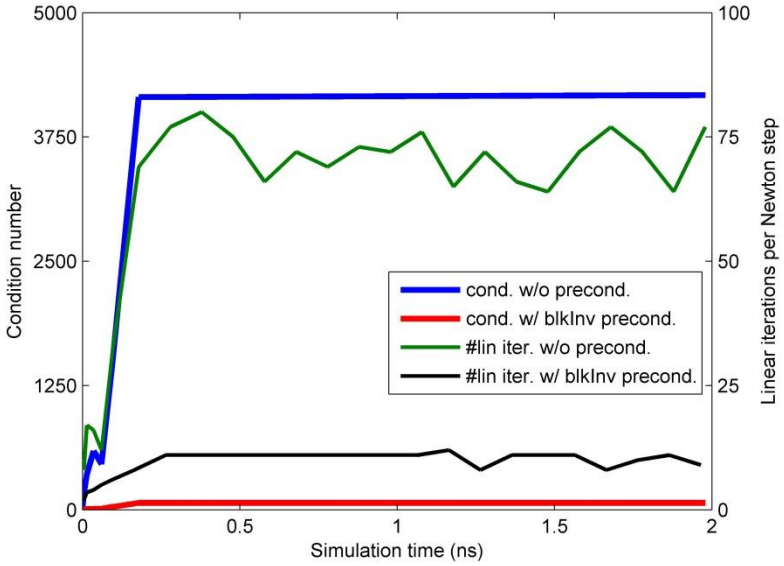

Figure 4.4: Condition number and the number of linear iteration per Newton iteration with and without block-diagonal preconditioning.

We also experimented with the optimal performance of iLU for this problem. The parameters we tuned include the iLU methods (iLU0 and iLUt), the max-fill bandwidth of the inverted matrix by iLUt (1, 3, 10, 30, 50). In the block-diagonal preconditioner the

numerical experiments included varying the distance threshold (1 pm, 1nm, 3nm, 5nm, 10nm, 20nm). For all methods, we experimented with the frequency of updating the coefficient matrix (every {3, 10, 30, 50} time steps), and using the left or right preconditioners. The best, worst, and mean performance with all the options is summarized in Fig. 4.5. The block-diagonal method achieved 3.9x acceleration while the best iLU method gives 2.9x speed-up. However, it is hard to predict which iLU method is proper to use before the simulation starts. As indicated by Fig. 4.5, the simulation could be slowed down by as much as 22x if iLUt1 (iLUt, max-fill bandwidth is 1) is chosen. In other words, the performance of the iLU methods is unstable. To the contrary, choosing the parameters for the block-diagonal method is fairly straightforward. It delivers satisfying performance as long as the distance threshold is set to be greater than the thin-film thickness.

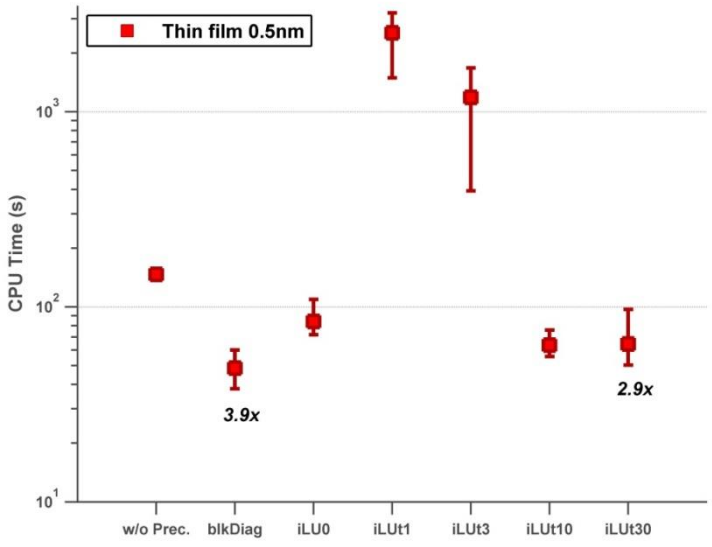

Figure 4.5: The lowest, highest and average CPU time with different preconditioning methods. The best achievable speed-up with the block-diagonal method and the iLU method is tagged.

Table 4.1 summarize the optimal simulation results achieved by the implicit method without preconditioning, the implicit method with the iLU preconditioning, and with the block-diagonal preconditioning. The CPU time for the preconditioners accounts for the coefficient matrix factorization operations and the sparse matrix-vector product operations to apply the preconditioning. Without preconditioners, the evaluation of the magnetostatic field on GPU only takes a small portion of overall time (2.1%), while the iterative linear solver takes 96.3%. The magnetostatic field evaluation is subdominant here because (a) the problem scale is small and (b) the numerical problem is very stiff. Therefore, preconditioning is necessary in this case to accelerate the simulation by reducing the number of linear iterations. From Table 4.1, we find that both preconditioning methods efficiently reduce the number of linear iterations. However, the block-diagonal method results in a better CPU time performance due to its low cost per evaluation. The time spent on the block-diagonal method is only 13.5% of the total CPU time, which is significantly lower than 40.4% for the iLU method. As a result, the block-diagonal method achieved 3.9x acceleration while the iLU method only gives 2.9x.

Table 4.1: Summary of statistical data for different preconditioning schemes.

	Implicit Method	Implicit Method w/ ilu Precond.*	Implicit Method w/ blkDiag Precond.**
Total T_{CPU} (s)	146.6	57.1	38.1
Avg. T_{CPU} per time step (ms)	268.4	103.1	63.0
Avg. T_{CPU} for preconditioner per time step (ms)	NA	41.6	8.5
Avg. TimeStep (ps)	9.15	9.02	8.28
#Rhs Evaluation	574	900	732
Avg. #linear iterations per Newton step	71.43	3.70	6.72

* The best performance of all iLU methods tested is given by iLUt with max-fill bandwidth=30, coefficient matrix updated every 10 time steps, and with left preconditioning.

** The best performance of all block-diagonal preconditioning tested is given with distance threshold=10nm, coefficient matrix updated every 10 time steps, and with left preconditioning.

B. Magnetic Cone

Switching a soft magnetic cone is a good example for small simulations that have severe stiffness issue due to clustered mesh nodes. As depicted by Fig. 4.6, here we tested a cone that has a diameter of 400 nm and a height of 400 nm. The edge length is as small as 0.5 nm at the tip of the cone (50 nm in height) while ~20 nm elsewhere. The magnetic properties are A_{ex} = 1.0e-6 erg/cm, M_s = 1000 emu/cc, $H_k = 2K/M_s = 0$, $\alpha = 0.2$. Applied field $\mathbf{H}_{app}$=(0, 0, 1.0e3) Oe is added to switch the uniform initial magnetization from **m** = (0, 0, -1) to **m** = (0, 0, 1). The number of nodes and elements in the mesh are 3.5K and 17K, respectively. The simulation time was 1 ns with absolute and relative time step tolerance of 1e-4, and linear iteration tolerance of 5e-2.

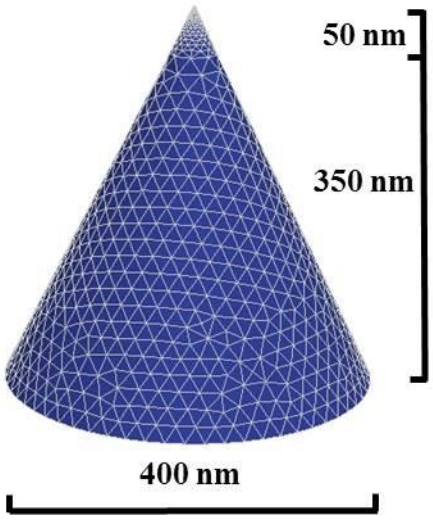

Figure 4.6: Geometry size and the mesh of the tested magnetic cone.

From Fig. 4.7 we can see that the condition number of the block-diagonal preconditioned matrix is always lower than the original matrix. The number of linear iterations per Newton step is also much smaller than in the original case. The correlation of the condition number and the number of linear iterations per Newton step is obvious.

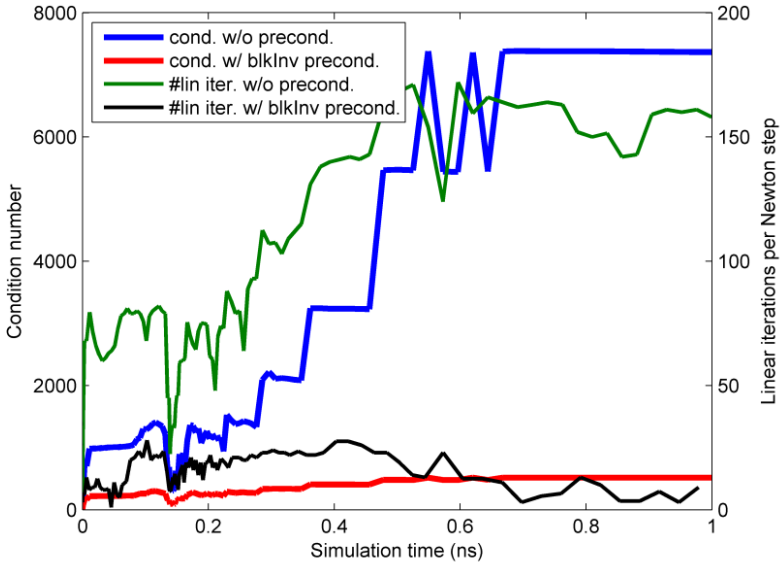

Figure 4.7: Condition number and the number of linear iteration per Newton iteration with and without block-diagonal preconditioning.

Again, numerical experiments were performed for the best performance of different options, similar to the thin film example. The best, worst, and mean performance with all the options is summarized in Fig. 4.8. The speed of simulation can be greatly improved by proper preconditioners (up to 5.9x by the block-diagonal method). However, unsuited preconditioner could slow down the simulation by a factor of 1.6x (iLUt3). Fig. 4.8 also indicates that the performance of the solver utilizing the block-diagonal preconditioner is stable with various thresholds and is better than the ones using iLU preconditioners. Thus, we can simply choose a certain threshold by the block-diagonal preconditioner, for example, 1pm, which will give a close-to-best performance.

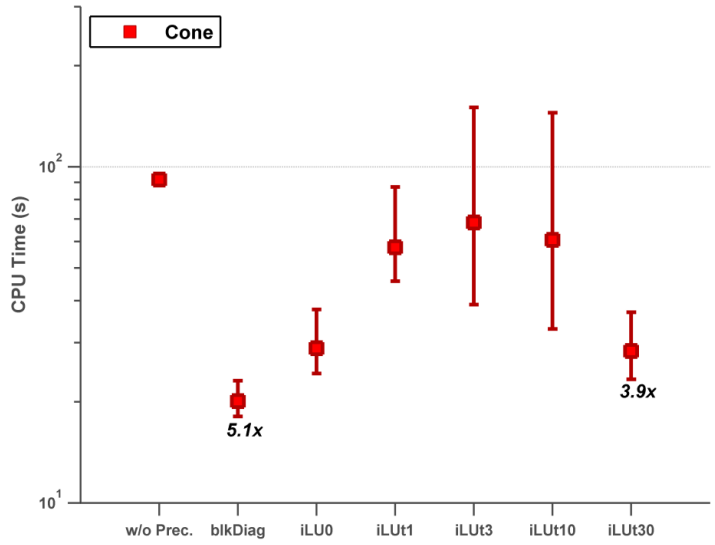

Figure 4.8: The lowest, highest and average CPU time with different preconditioning methods. The best achievable speed-up with the block-diagonal method and the iLU method is tagged.

The numerical metrics of the simulations demonstrated are summarized in Table 4.2. Again, due to the small problem size, the computation of magnetostatic field on GPU only takes a very small portion of the overall time (1.3% without preconditioning), while

the iterative linear solver takes most of the simulation time (77.1% without preconditioning). Preconditioning is necessary to accelerate the simulation by reducing the number of linear iterations. We can learn from Table 4.2 that the preconditioning quality by the iLU preconditioner, in this case, is better than the block-diagonal preconditioner. Specifically, the iLU method reduced the linear iterations by 24.8x while the block-diagonal method gives 7.0x. However, the block-diagonal preconditioner is still the preferable option considering the computational time, because the preconditioner itself is very fast. The percentage of CPU time spent on preconditioning is 7.3% by the block-diagonal preconditioner, comparing with 42.2% by the iLU preconditioner.

Table 4.2: Summary of statistical data for different preconditioning schemes.

	Implicit Method	Implicit Method w/ ilu Precond.*	Implicit Method w/ blkDiag Precond.**
Total T_{CPU} (s)	91.7	23.3	18.1
Avg. T_{CPU} per time step (ms)	172.7	43.1	35.8
Avg. T_{CPU} for preconditioner per time step (ms)	NA	18.2	2.6
Avg. TimeStep (fs)	1.88	1.84	1.98
#Rhs Evaluation	687	741	744
Avg. #linear iterations per Newton step	91.68	3.70	13.11

* The best performance of all iLU methods tested is given by iLUt with max-fill bandwidth=30, coefficient matrix updated every 30 time steps, and with left preconditioning.

** The best performance of all block-diagonal preconditioning tested is given with distance threshold=1nm, coefficient matrix updated every 10 time steps, and with left preconditioning.

C. Magnetic Write Heads

The magnetic write head is one of the common applications of the Finite Element method based micromagnetic solvers. To achieve proper accuracy, the number of unknowns, which is proportional to the number of nodes in the mesh, can be very large. We conducted a set of simulations with the same magnetic write head but with different meshing methods, leading to the number of mesh-nodes as 265K (write head I), 750K (write head II) and 4M (write head III). The three examples are representative in that they cover different problem sizes and mesh qualities. The mesh quality varies but it does not necessarily depend on the mesh size. Among the models of concern, the 750K case has the best mesh quality while the other two models have similar lower mesh quality. The good quality here refers to the mesh containing fewer tetrahedron elements with small angles. Its strong effects on the stiffness of the coefficient matrix are reflected in the number of linear iterations per Newton step.

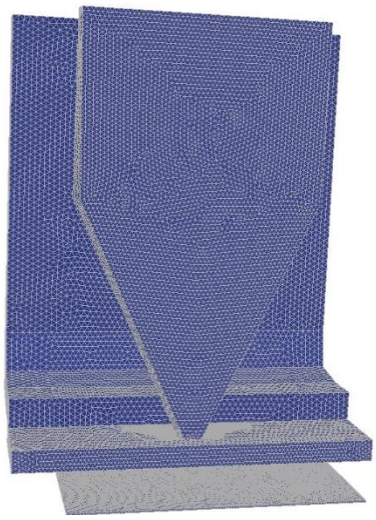

Figure 4.9: Geometry and the mesh of the tested magnetic write head.

The geometry size of the write head in Fig. 4.9 is {6, 4.71, 7.36} μm in the x, y and z direction, respectively. The magnetic properties of the model are summarized in Table

4.3. The initial magnetization is **m** = (1, 0, 0). The periodic applied field generated by a coil surrounding the main pole is added to switch the magnetization at the tip. The simulation was run for 2 ns with absolute and relative time step tolerance of 1e-4, and linear iteration tolerance of 5e-2. The maximal Krylov subspace size, i.e. the maximal number of iterations, was set so that the linear system always satisfy the tolerance before reaching the limitation.

Table 4.3: Summary of magnetic properties in the tested write head.

	Easy Axis	K (erg/cc)	M_s (emu/cc)	A_{ex} (erg/cm)	α
Pole tip	(1 0 0)	1403	2.35	1.1e-11	0.2
SUL	(1 0 0)	955	1.6	1.1e-11	0.2
Other parts	(1 0 0)	3980	1.0	1.1e-11	0.2

The condition number and the number of linear iterations per Newton step result for the write head I are shown in Fig. 4.10. The fluctuation of the original (un-preconditioned) curves can be explained as follows: (1) the ODE solver is initialized with a small time-step; (2) The time step increases during 0ns ~ 1ns, and 1ns ~ 2ns when the applied coil field is relatively stable, leading to relatively large condition number; (3) The phase of the applied coil field changes at 1ns, inducing dynamics into the system. As a result, the time step is adaptively reduced and the condition number of the coefficient matrix is relatively small. A clear correlation between the condition number and the number of linear iterations per Newton step is demonstrated by the non-preconditioned curves. On the other hand, the block-diagonal preconditioning method shows reliable conditioning quality, resulting in small condition number and the number of linear iterations per Newton step.

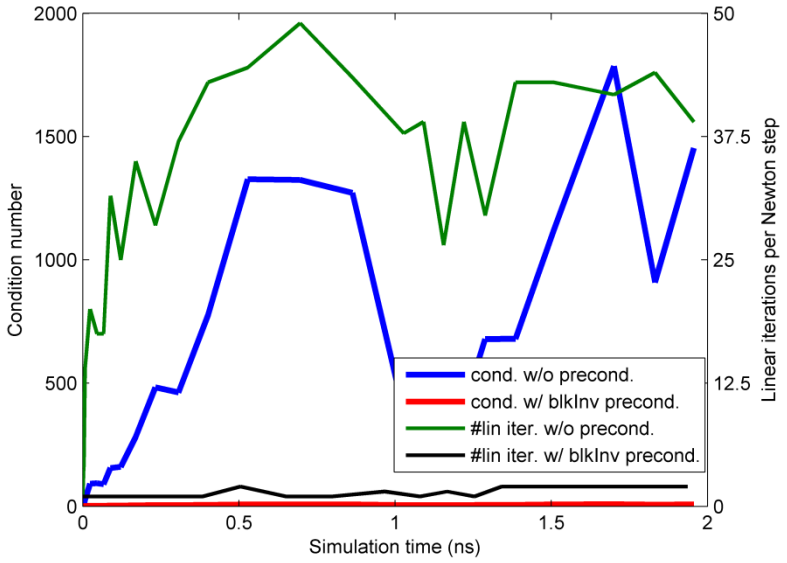

Figure 4.10: Condition number and the number of linear iteration per Newton iteration with and without block-diagonal preconditioning in the test case write head I (250K).

The same exhaustive search for the best performance for different options is conducted as introduced in the previous (film and cone) examples as summarized in Fig. 4.11. We see a significant speed-up by proper preconditioner utilization (up to 6.4x with the block-diagonal preconditioner). The speed-up achieved by the block-diagonal method is better than the other preconditioning method in all three test cases. It is also apparent from Fig. 4.11 that the results from the block-diagonal preconditioner have much smaller deviation than other methods.

Despite the 15x problem size difference between the test cases “write head I” and “write head III”, both have normal mesh quality which leads to similar averaged linear iterations per Newton step (~33). In these two cases, without preconditioning about 90% of the CPU time is spent on the linear iterative solver. With preconditioning, the CPU time spent on operations other than the preconditioner is roughly the same. In such

scenario, the fast speed of block-diagonal preconditioner is favorable, as indicated by Fig. 4.11.

"Write head II" has a very good mesh quality, such that only 49.1% of the total CPU time is spent on the linear solver when no preconditioner is applied. As a result, even though the problem size of "write head II" is 3x larger than "write head I", the speed without preconditioning is faster. In this case, exploiting the preconditioners has to be careful because a bad preconditioner could generate worse results, as shown by the iLU0 results in Fig. 4.11. However, the block-diagonal preconditioner can still improve the speed of simulation by 1.3x, which demonstrates its robustness.

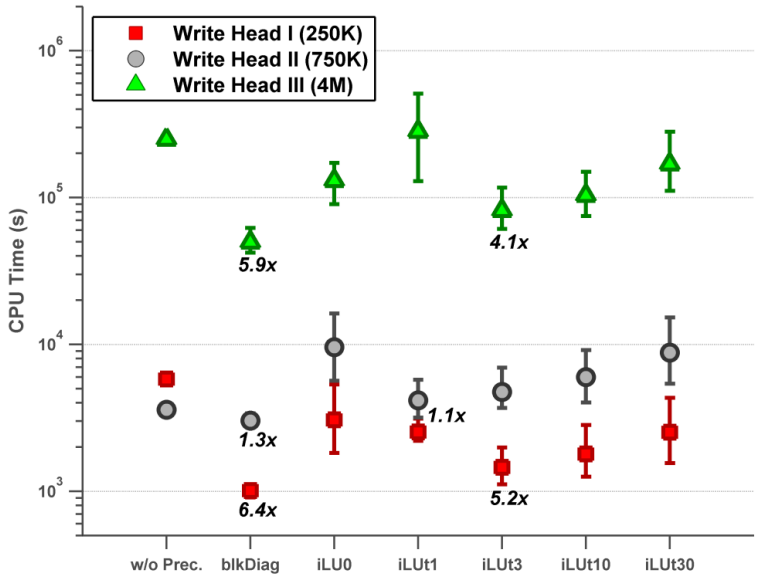

Figure 4.11: The lowest, highest and average CPU time with different preconditioning methods on three write head test cases, respectively.

The best results of the three write head test cases are summarized in Table 4.4. The block-diagonal preconditioner is robust in that it always reduces the number of linear iterations per Newton step. The CPU time spent on the block-diagonal preconditioner is

about 10% of the total CPU time. The preconditioner itself is sufficiently "light" so that even when the coefficient matrix is not very stiff (write head II), it still leads to the performance improvement. The iLU method is also efficient in reducing the number of linear iterations when a specific optimal parameters are chosen. However, there are large fluctuations in the performance depending on the parameter choice, which makes it hard to use. For the block-diagonal preconditioner, we can simply choose a default small threshold for selecting blocks to result only in 3 x 3 blocks, still achieving an improved performance. If the stiffness source is known, the threshold can be related to that source (such as in the thin film case) achieving an even better performance.

Table 4.4: Summary of statistical data for different preconditioning schemes.

	Write Head I (265K nodes, 1.4M elements)			Write Head II (750K nodes, 4M elements)			Write Head III (4M nodes, 22M elements)		
	Implicit Method	Implicit Method w/ ilu Precond. *	Implicit Method w/ blkDiag Precond. **	Implicit Method	Implicit Method w/ ilu Precond. *	Implicit Method w/ blkDiag Precond.**	Implicit Method	Implicit Method w/ ilu Precond. *	Implicit Method w/ blkDiag Precond. **
Total T_{CPU} (s)	5.8e3	1.1e3	9.1e2	3.6e3	3.2e3	2.7e3	2.5e5	6.1e4	4.2e4
Avg. T_{CPU} per time step (s)	4.76	0.88	0.77	3.07	2.76	2.27	195.3	47.81	32.33
Avg. T_{CPU} for precond. per time step (s)	NA	0.17	0.08	NA	0.51	0.18	NA	10.93	3.34
Avg. TimeStep (ps)	1.64	1.57	1.72	1.72	1.74	1.66	1.57	1.56	1.53
#Rhs Evaluation	1949	1926	1913	1989	1822	1888	2194	2260	2237
Avg. #linear iterations per Newton step	30.96	1.49	1.26	3.79	1.79	1.54	35.94	4.02	3.57

* The best performance of all iLU methods (iLU0 and iLUt) tested on write head {I, II, III} are given by iLUt with max-fill bandwidth = {3, 1, 3}, coefficient matrix updated every {50, 30, 30} time steps, and with right preconditioning.

** The best performance of all the block-diagonal preconditioning tested on write head {I, II, III} are given with distance threshold = {10nm, 3nm, 1nm}, coefficient matrix updated every 10 time steps, and with right preconditioning.

D. Matrix Factorization Speed Scaling

One important advantage of the block-diagonal method over the iLU method is the speed scaling of matrix factorization over the problem size, which is demonstrated by Fig. 4.12. Here we compare the averaged CPU time spent on each factorization operation by the iLUt methods and the fully isolated block-diagonal method as an example. The data comes from the presented four test cases. (The thin film test is omitted since the size is similar to the magnetic cone test.) It is obvious that the speed of the block-diagonal factorization scales linearly while the iLUt factorization speed scales with a higher order. This indicates that the block-diagonal method could be even more favorable when simulating much larger problems, e.g. problems with over 10M mesh nodes.

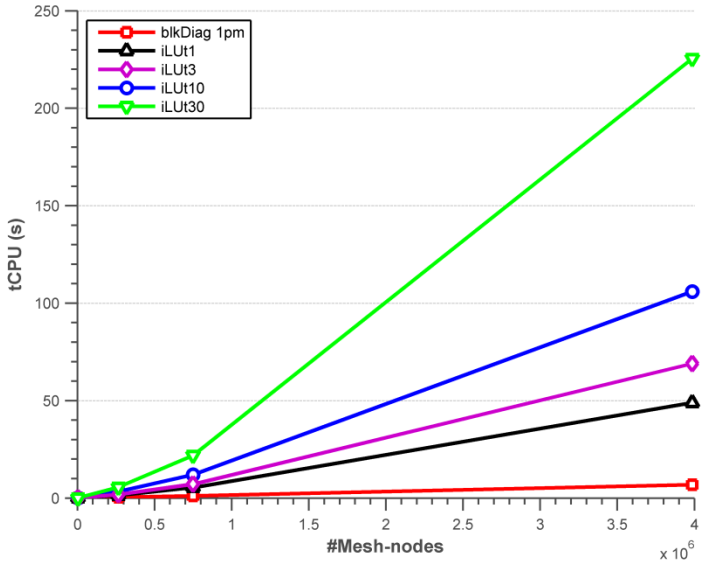

Figure 4.12: Speed scaling of matrix factorization against problem size.

In summary, a new block-diagonal preconditioning method for Jacobian Newton-Krylov approach in the context of BDF solving time integration of micromagnetic simulations is proposed. The formulation of the method was explained in detail and a

variety numerical examples demonstrated its speed, stability, and scalability. The time spent on preconditioning is much less than the time for the iLU method, so it is a good candidate for simulations that cannot achieve good performance due to slow preconditioner factorization. The convenience of usage is indicated by the robust high performance of the block-diagonal method. The method is highly parallelizable, thus it is ready to be imported to multi-core CPU and GPU platforms.

Chapter 4, in part, is currently being prepared for submission for publication of the material, where the dissertation author was the primary investigator and author of this paper: S. Fu, R. Chang, I. Volvach, M. Kuteifan, S. Couture, M. Menarini, V. Lomakin, "*Block Diagonal preconditioner for implicit time integration in finite element micromagnetic solvers*".

5. Micromagnetic simulations of Advanced Magnetic Media and Recording Systems

We start with discussing the magnetic recording system, which is composed of a magnetic reader/sensor, magnetic write head, and magnetic media. Each component of the system requires intense research and manufacturing efforts to demonstrate the modern high-capacity hard disk drive. The focus of this chapter is on the numerical simulations for the magnetic recording media, the structure and material of which have fundamentally changed over the past 20 years.

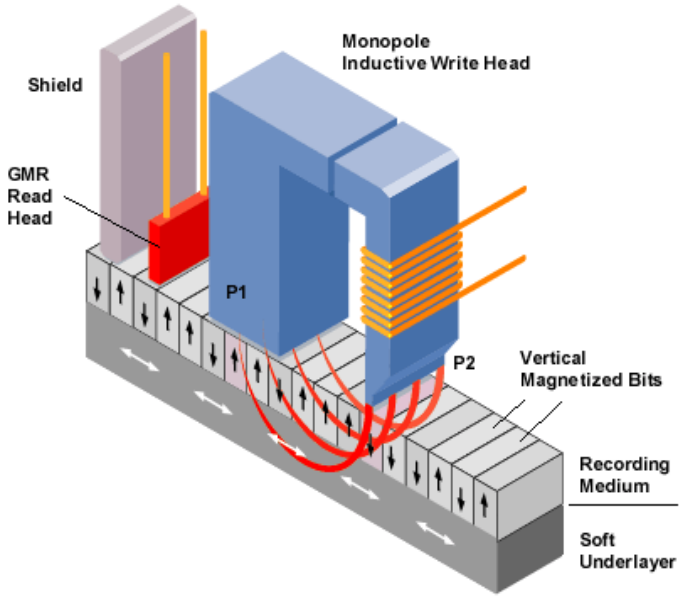

Figure 5.1: Magnetic recording system.

5.1. A Brief History of Magnetic Media and Recording Systems

Areal density has been quoted as the key factor to mark the progress of the magnetic recording. The areal density of the hard drive has increased from 20 GB/in^2 in the year 2000 [87] to 1.3 TB/in^2 in the year 2015 [88]. The technology of perpendicular magnetic recorder (PMR) [89], shingled magnetic recording (SMR) [90] has greatly contributed to such as amazing areal density speed increase. Currently, the PMR technology is dominating the market. The hard disk drives shipped with the SMR technology is mainly used in the cold storage, due to the fact that writing shingled tracks takes much longer time than the normal PMR tracks. However, the speed of the areal density increase has slowed down during the past few years [88].

The limiting factor of the continuous high-speed areal density increase is the thermal stability in the magnetic media [87]. The magnetic media grain volume has to decrease to improve the areal density in the traditional magnetic media. However, the information stored in the magnetic media tends to be unstable when the volume of the magnetic grain decreases. To explain this, we consider a simple magnetic grain model where only uniaxial magneto-crystalline and Zeeman energy are considered. According to the Neel-Arrhenius formula [91]

$$\tau = f_0^{-1} \exp\left(E_b / k_B T\right), \tag{5.1}$$

where τ is the mean exit time, namely, the averaged magnetic grain stabilization time, E_b is the energy barrier

$$E_b \sim K_u V\left(1-H/H_0\right)^n, \tag{5.2}$$

k_B is Boltzmann constant, T is temperature, f_0 can be defined with an analytical asymptotic formula [91]:

$$f_0 = 2\alpha_Q \gamma H_K \sqrt{K_u V/k_B T}\sqrt{4/\pi}\left(1-H/H_k\right)^2, \tag{5.3}$$

where α_Q is a dimensionless damping parameter, γ is the electron gyromagnetic ratio, K_u is the uniaxial anisotropy constant, $H_K = 2K_u/M_s$, M_s is saturation magnetization, V is the grain volume and H is the applied field on the magnetic grain. From Eq. (5.1) , Eq. (5.2) and Eq. (5.3), we see that the thermal stability of the magnetic material is exponentially related to the volume of the grain size V and uniaxial anisotropy constant K_u.

An obvious approach to improving the thermal stability of the magnetic materials is to maximize the magnetocrystalline energy density, which is related to material coercivity K_u [87]. Nevertheless, large coercivity K_u would decrease the writeability of the magnetic materials.

Intense efforts have been made to explore novel methods of magnetic recording, including Heat Assisted Magnetic Recording (HAMR) [92], Microwave Magnetic Recording (MAMR) [93], Two Dimensional Magnetic Recording (TDMR) [94], and Bit Patterned Media (BPM) [95]. HAMR and MAMR are two promising energy-assisted magnetic recording methods. HAMR utilize a laser on the high-coercivity magnetic material to decrease the effective coercivity within a confined spot so that the data can be written at this location. MAMR takes advantage of the microwave generated by a STO to

periodically pump energy into the magnetic recording system so that the magnetization could be switched on a high coercivity material. TDMR focuses more on the signal processing of the reading process, which makes use of an array of heads to read the magnetization simultaneously. The technology is referred as 2D since a single track cannot be read or written successfully without considering the adjacent tracks [96]. BPM is proposed to use engineered well-defined magnetic islands so that each island stores exactly one bit of data [94].

Although hard disk drive shipped with HAMR or TDMR technology is predicted to roll out in 2017-2018, the evolution of the magnetic recording technology has been postponed by a few years. Difficulties of applying the new techniques come in various ways. For instance, the high temperature created by the laser in HAMR technology may degrade the surface of the magnetic recording materials and even the magnetic heads. BPM required highly uniform processing of the materials over an entire disk which might greatly increase the cost the manufacturing, meanwhile the requirement on the carefully synchronized reading/writing process with the islands positions is also very challenging. TDMR and MAMR have not been proven to be able to deliver noticeable performance increase yet.

A roadmap figure at the Advanced Storage Technology Consortium (ASTC) summarized the novel technologies and the challenges, as shown in Fig. 5.2 [97]. With combined HAMR and BPM, it is promising to achieve 10 Tb/in^2 areal density in 2025, which is 10x denser than the current hard disk drives. However, progress has to be made. Numerical modeling and simulations play an important role in predicting the capabilities of the proposed novel technology configurations. In the following sections, with the

currently dominating technology PMR being addressed, a modeling approach designed for the latest magnetic media design is introduced. The numerical results are compared with the traditional modeling results to address the importance of the new model.

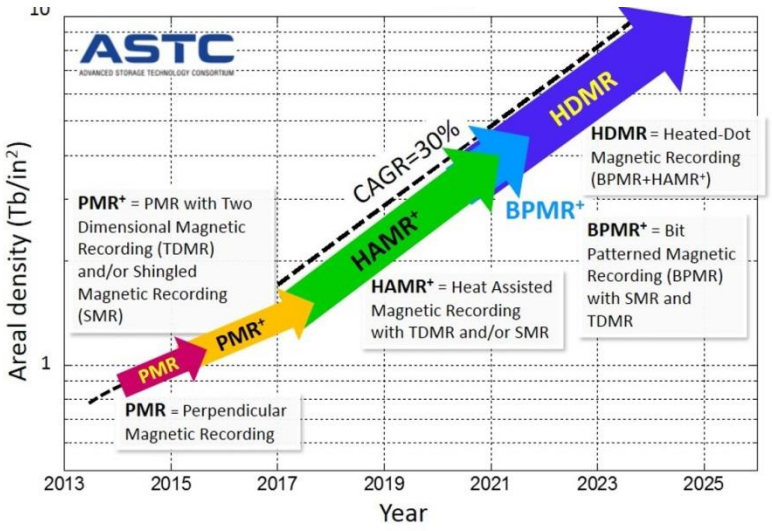

Figure 5.2: Roadmap of hard disk drive technology [97].

5.2. Perpendicular Magnetic Recording

In this section, we discuss more technical details of PMR [98] since it is the currently dominating technology.

Compared with the previous generation, namely the longitudinal magnetic recording, PMR uses magnetic media with perpendicular anisotropy. Thus the stable state of the magnetization in the media is out of plane. In PMR, a soft under-layer (SUL) is used to guide the write-head field through the magnetic media layers, so that the effective writing field is stronger. The geometrical property of the write-head is designed to utilize a monopole shape so that the strength of the field is the strongest when penetrating the media. The magnetic media used in PMR can be thicker to deliver better thermal stability

and promote the magnetostatic field to stabilize the relaxed magnetization. Moreover, the coercivity (K_u) and the saturation magnetization (M_s) also can be stronger [99]. As a result, the thermal stability is further improved to achieve higher areal density.

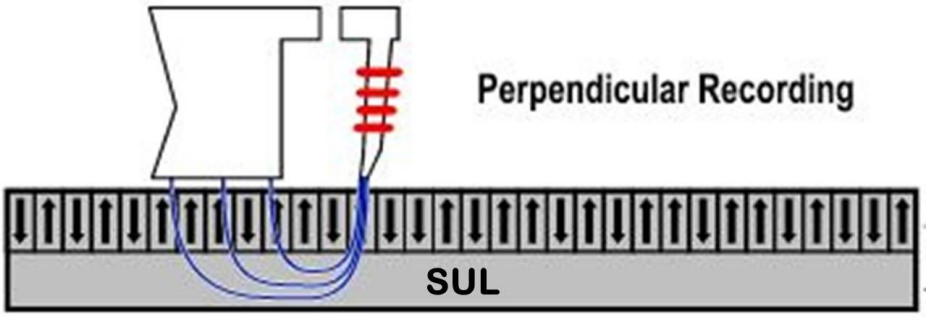

Figure 5.3: Schematic diagram of perpendicular magnetic recording [100]**.**

More media layers have been added to assist the reversal process during the magnetic recording, which is referred as Exchange Coupled Composite (ECC) media [101][102]. ECC media has been essential in achieving improved recording performance [103] as both areal and linear density are increased without compromising the thermal stability.

Figure 5.4: Schematic diagram of the ECC media. Between the cap layer and the granular layers (oxide layers) there are the exchange-coupled layers.

ECC media contains an exchange coupled cap layer and granular oxide layers, as shown in Fig. 5.4. Most of the lateral exchange in the media arises from the cap layer and the noise performance of media strongly depends on the cap layer thickness. Thus, optimizing the cap layer is essential to achieve good recording performance. Currently, the short bit length in a code word (the 1T bit length) is approaching the media grain diameter and non-uniform magnetization behavior in the cap layer may have important effects. Therefore, it is important to model non-uniform behavior in the cap layer and in thin oxide layers to accurately evaluate recording performance. In the following section, two new models are proposed to account for the non-uniform behavior that comes with the modern cap layer in the ECC media.

5.3. Discretized Cap Model of Magnetic Media

The traditional macro-spin model [103] assumes the magnetization and material parameters such as saturated magnetization (M_s), and anisotropy field (H_k) can be modeled as being uniform in each layer comprising a grain. In addition, the exchange interaction (J_s) within a grain is not included in the model. However, scattering experiments [104][105] show a radial grading of media properties in each grain. In the cap layer of current media, magnetization may vary gradually within grains to form closure domains. The macro-spin model may not be able to fit the semi-continuous nature of the cap layer and the switching properties of very thin magnetic layers often found in the current generation of magnetic media. As a result, performance metrics based on a macro-spin model may diverge from the experimental data at high linear densities in

many key areas, such as resolution, non-linear effects [106], and particularly signal-to-noise-ratio (SNR).

In this section, we report simulation results from two discretized models: Model 1 discretizes all regions in a granular/semi-continuous layer, which includes both the grain centers and the boundaries, and Model 2 discretizes only the grain centers. Both models employ a distribution of material parameters across each grain. To efficiently implement both schemes we developed the models to run on a cluster of GPUs. A detailed description of the models is provided. We also compare the performance of the fully discretized cap models for ECC media to the macro-spin model in this section.

5.3.1. Discretized Cap Layer Modeling

As shown in Fig. 5.5, our macro-spin model subdivides the grains and grain boundaries into cubic cells. The fields in each cell, with the exception of internal exchange, are computed independently during the calculation of the effective field (H_{eff}) in the LLG equation which also includes a thermal field. Only the boundary cells are subject to a lateral exchange interaction. The fields are then averaged in each granular layer. A single spin is then used in the LLG equation. Thus, the magnetization of the cubic cells inside a single grain are all aligned. The physical properties are assumed to be uniform across the entire grain. Thus, the field operating on each layer represents an effective field which includes the effects of internal interactions in an average sense. The anisotropy field employed in a macro-spin model may, therefore, be very different from the anisotropy field in a discretized media model.

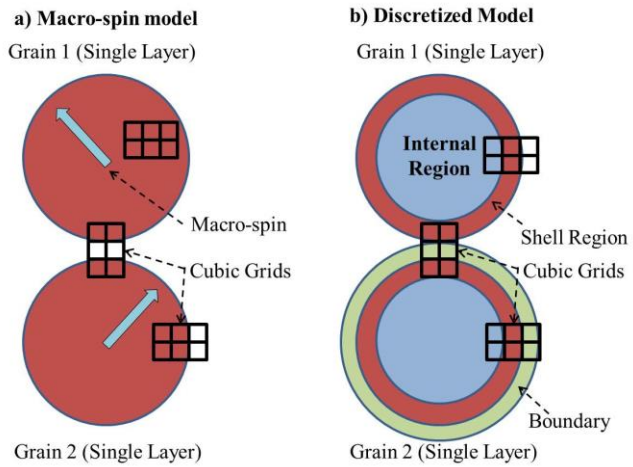

Figure 5.5: Schematic comparison of a) macro-spin model with b) discretized model.

In contrast to the macro-spin model, a discretized media model allows each cell to have distinct properties such as M_s, H_K, and J_s. This approach provides a better approximation to the cap and thin oxide layers, where the properties are thought to be graded both laterally and vertically. The influence of the inter-layer and inter-granular exchange strength on the magnetic recording performance has been often discussed in the literature [102][107]. It is found that if one exceeds an optimal exchange strength, increased cluster size in the media results in lower signal-to-noise ratio (SNR). Increased cap inter-granular exchange can also result in the lower resolution [103]. Exchange also influences the writeability of the media, which can be defined as the percentage of the grains that are switched in the presence of an external magneto-static field [103]. Both inter-granular and inter-layer exchange coupling have a large impact on writeability. The exchange field generated by the cap layer increases with M_s in the cap layer and decreases with M_s in the oxide layers [102]. Given the importance of these magnetic properties, an accurate description of the distribution of these parameters may be essential to reliably assessing recording performance.

In a discretized media model, the magnetization of each discretization cell is an independent variable in the LLG equations. The smaller length scale makes describing closure domains in the cap layer possible. The influence of these domains is reflected in the degradation of the read-back resolution and in high-density SNR performance in magnetic recording experiments. Fig. 5.6 illustrates how a discretized model can describe closure domains at transitions while the macro-spin model cannot. The same figure compares plots of the simulated in-plane magnetization for a small bit pattern, 6T-6T-2T-1T-2T. It shows the appearance of closure domains in the dual-layer discretized cap model, while the domain wall generated by macro-spin model is not significant. The formulation of these closure domains leads to a poorer 2T/6T resolution than found in the macro-spin model.

In this study, we only discretize the cap layers but our implementation is general so that any media layer can either be discretized or treated as a macro-spin.

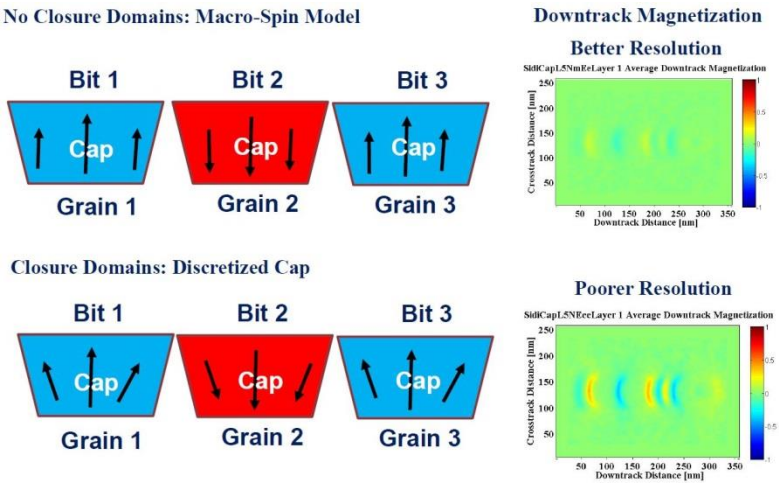

Figure 5.6: Schematic view and simulation results showing how closure domains can be formulated in the discretized model.

5.3.2. Discretized Cap Layer Models

Two discretization models are investigated and we compare the simulation results to the commonly used macro-spin model. In the first model, the space between the grains is considered magnetic and the inter-granular exchange between the grains results from the interactions between the boundary cells. In the second model, the boundary between the grains is non-magnetic and inter-granular exchange is determined by a surface interaction. These two models are illustrated in Fig. 5.7.

In order to demonstrate both models, we implemented a general discretization scheme, where grain boundaries may possess magnetic properties. Properties can be varied from layer to layer so the model can describe layer properties, in particular, cap layer properties, which can change with layer thickness. The magnetic properties within a grain vary as a function of radius [104][105]. We incorporate this effect by partitioning the grains into three regions: an internal region, a shell region and a boundary region. Each region is assigned different magnetic properties (M_s, H_k and J_s).

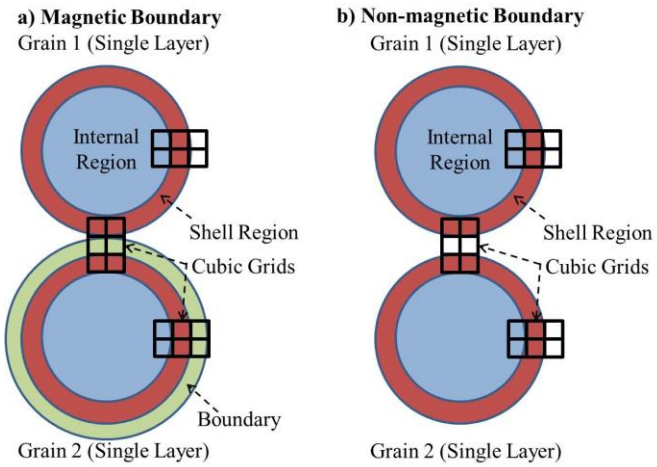

Figure 5.7: Schematic comparison of a) magnetic boundary with b) non-magnetic boundary in the discretized model.

5.3.3. GPUs Implementation

A fully discretized model increases the computational effort needed to simulate the recording process as both the total number of variables in the problem increases and the LLG time-step must be reduced due to the large exchange fields present in the model. In this work, we used the Heun method and the time-step had to be reduced by a factor of 10 to maintain stability. To overcome these difficulties, the micromagnetic media model was implemented on a massively parallel computational platform composed of GPGPUs [16][6].

We used Nvidia Geforce Titan GPUs, which have 2688 streaming processors. Titans have a global memory of 6 GB, which is sufficient for the simulation size of 4096 pseudo-random-bits (PRBS).

In our micromagnetic model, each cubic cell is assigned a CUDA thread/processor. The full computational workload is offloaded to GPU so that no GPU-CPU memory transfer is involved during the computations. The most computational-intensive component of the model is the evaluation of the magneto-static field. These computations are accomplished using FFTs, which are accelerated by the Nvidia cuFFT library. The computation of other fields and the time-integration are primarily local operations, which are parallelized based on one-thread-per-observer strategy on the GPU.

5.3.4. Numerical Results

As depicted in Fig. 5.8 (a), a four-layer ECC media is used in the simulations, where the cap layer was modeled as two discretized layers to approximate vertically graded material properties.

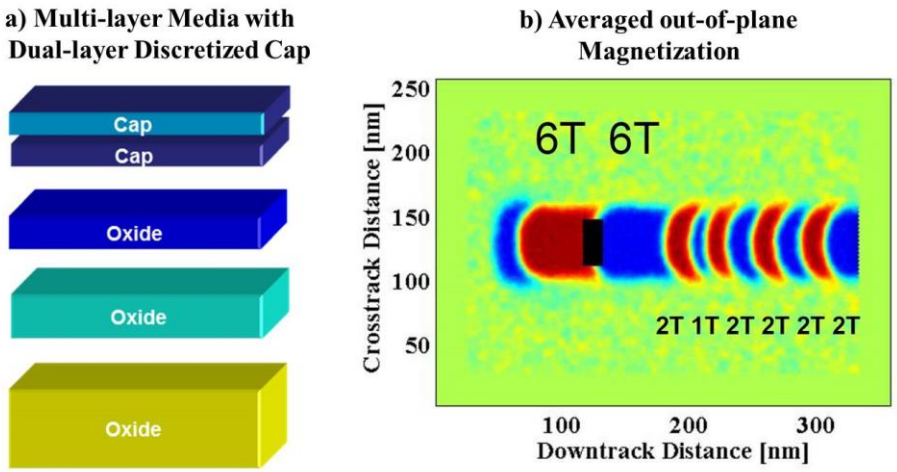

Figure 5.8: a) Layout of the magnetic layers in the ECC media that is simulated; b) Bit pattern of the magnetization used in the simulation.

The bit pattern used to generate all the test results is shown in Fig. 5.8 (b). It contains long magnetic bits (6T) from which Jitter and DC SNR are obtained. The Jitter is computed as the variation of the transition position between the two 6T bits. DC SNR is determined at the center of the first 6T bit. The Magnetic Writing Width (MWW) is measured at several cross-track positions in the center of the 6T magnets and then averaged.

The bit pattern also contains short magnetic bits (2T and 1T), from which resolution and a high-frequency SNR are determined. A “Contextual SNR” is measured from the 2T-1T-2T regions of the bit pattern as shown in Fig. 5.9. The Contextual Signal is determined as the difference in signal at the center of the 1T bit and the signal at the center of the second 2T bit. The noise is the normalized standard deviation of the signal at all points within the entire region defined in Fig. 5.9. The 2T pattern is repeated so that the 2T/6T resolution results can be extracted from a central 2T bit. Here 2T/6T resolution is computed as the ratio of the peak amplitude at the 2T bit to that at the 6T bit.

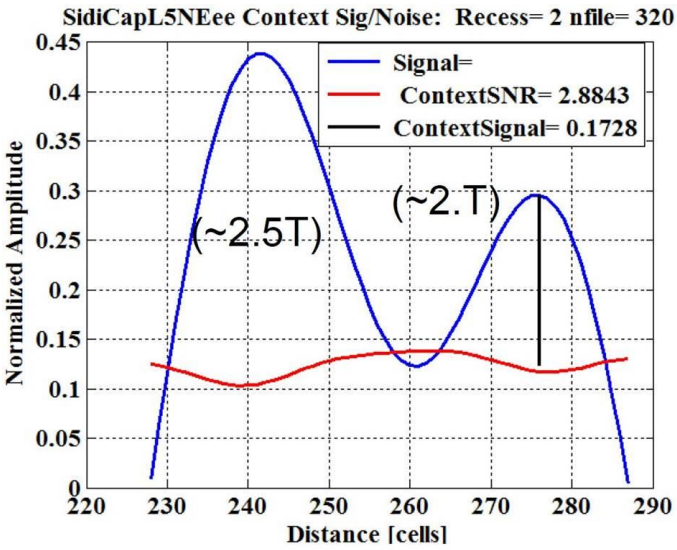

Figure 5.9: The region where Contextual SNR is measured and an illustration of how it is measured.

A. Low-Frequency Parameter Performance

In Fig. 5.10, we show the 2T/6T resolution extracted from a discretized cap simulation compared to experimental results. Very good agreement between the simulations and experiment is seen. In these simulations, we employed a standard FEM model of the writer and reader used to represent a current server class drive product. Under similar circumstances, the macro-spin model has 4%-5% better resolution than the discretized model. The difference comes from the fact that the macro-spin model is not capable of describing closure domain in the cap layer.

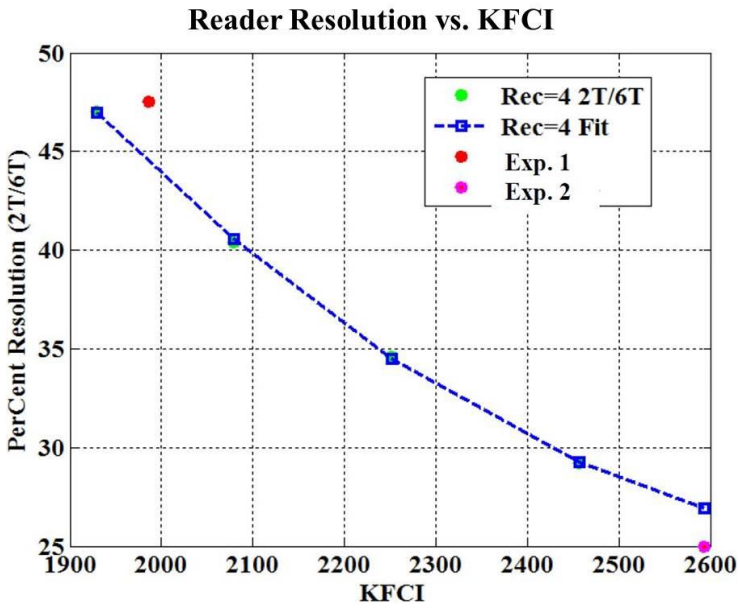

Figure 5.10: Comparison of simulation to the experiment: Resolution versus KFCI (linear density).

B. Noise Performance

Resolution results can often be fitted by varying the magnetic spacing between the head and media. For a more thorough comparison of the discretized model and the macro-spin model we also investigate the high-frequency noise performance of the two methods. In this study, we use Contextual SNR as the high-frequency noise metric.

Here, we evaluate the equivalence of the models by matching the low-frequency parameters (jitter, resolution, MWW, and DC SNR), and compare the Contextual SNR produced by the models. Table 5.1 lists the simulation results based on the two discretized models and the macro-spin model. The low-frequency results are matched by tuning lateral and vertical exchange constant Js and Jv, Hk, and the reader recession height. We find that even if the Jitter matches at 2.02 nm, resolution matches at ~30%, MWW matches at ~63nm, and DC SNR matches at 25.8, there are still significant variations in the Contextual SNR results. The discretized model can more accurately

describe the inter-granular exchange strength, which controls the interplay between resolution, cluster size (noise effects) and writeability. These high-frequency effects are not simply duplicated in a macro-spin model.

The noise performance figures indicate that the discretized media model can produce quite different results from a macro-spin model. Combined with the low-frequency performance discussed in the previous section, we feel that the discretized model offer significant advantages over the macro-spin model in assessing current magnetic recording systems.

Table 5.1: Representative simulation results.

Models[a]	Jitter (nm)	Res. (%)	MWW (nm)	DC SNR (db)	Contex. SNR (db)
Macro-spin Model Run1	2.02	31.4	62.5	25.6	-0.24
Macro-spin Model Run2	2.02	28.4	62.9	25.5	-2.42
Discretized Cap: Mag. Grain Boundary	2.02	28.7	63.6	26.3	-0.84
Discretized Cap: Non-mag. Grain Boundary	2.07	29.2	63.9	25.8	-0.98

*The bit length = 10.34nm, grain pitch = 9.45nm and KFCI = 2465.

In summary, two micromagnetic models based on discretized media layers were developed. These models were compared to the macro-spin model and the results indicate that the discretized model offer significant advantages over the traditional model. In order to handle the additional computational workload necessitated by the discretized model, the code was implemented on a cluster of GPUs. The discretized media possesses a number of important features which can better describe resolution and high-frequency SNR as the bit length approaches the grain pitch in a magnetic recording system.

Chapter 5, in part, is a reprint but with minor modifications of the following journal articles, where the dissertation author was the primary investigator and author of this paper: S. Fu, L. Xu, V. Lomakin, A. Torabi, B. Lengsfield. "*Modeling perpendicular magnetic multilayered oxide media with discretized magnetic layers*." IEEE Transactions on Magnetics, vol. 51, no. 11, pp. 1-4, 2015.

6. Summary and Future Directions

6.1. Summary

The presented dissertation mainly contributed in the following three areas: 1) Fast field evaluation and time integration methods targeting the massively parallel system GPU, designed for the micromagnetic solvers; 2) Newly designed numerical model for modern magnetic recording media; 3) The design and implementation details of a micromagnetic solvers on various platforms, including newly emerged GPU systems.

Provided the basics of Micromagnetics, such as LLG equation, governing interactions in magnetic systems, and typical numerical modeling methods, a versatile micromagnetic solver FastMag is introduced. FastMag has been intensely used internally and externally to model complex magnetic structures.

The ability of FastMag simulating various magnetic structures highly relies on its strong computational capability, where the role of the massively parallel computational system GPU is indispensable. We provided a detailed description of GPU computing applied to micromagnetics, including the development progress of GPU during the past 10 years, its internal hardware model, and corresponding programming methods and tips. Use of GPUs on various platforms, such as desktops, clusters, and embedded systems was presented. The brief knowledge of GPU is one of the prerequisites of the understanding of the following chapters discussing the algorithms designed for GPU.

An efficient micromagnetic solver includes (1) highly optimized field/energy evaluation methods and (2) efficient and robust time integration methods. A time-consuming component in many micromagnetic solvers is the evaluation of the magnetostatic field/energy due to the fact that it requires a superposition of all unknowns. An efficient algorithm NUFFT was implemented to evaluate the magnetic potential from the effective magnetic charge densities. The algorithm is especially valuable for FEM solver, such as FastMag. Although the mathematical basics are the same as CPU algorithms, the implementation of GPU codes require careful management of the data arrangement and operation flows. Our NUFFT algorithm that runs on a relatively old GPU achieves up to 80x speed-up against the CPU implementation. Given the efficient implementation of NUFFT, the SPMV algorithm is the next bottleneck of the solver speed. An efficient implementation of the SPMV algorithm on single GPU and multiple GPUs were introduced. New implementation methods to save GPU memory in cases of ultra-large problems were presented. Two approaches to evaluate the magnetic potentials in the context of the FDM were also discussed.

A block-diagonal preconditioner was introduced to improve the speed and robustness of time integration of stiff micromagentic problems. This preconditioner has a low computational cost per evaluation, is robust for many micromagnetic problems, and achieves a good performance as compared to other more computationally intense preconditioners.

Finally, we presented new computational methods and tools for modeling modern magnetic recording media, based on multiple exchange coupled layers and a capping layer. The development of several novel technologies of the magnetic recording systems,

such as HAMR, MAMR, TDMR and BPM are briefly discussed and the role of the numerical simulation in such systems are addressed.

6.2. Future Directions

6.2.1. Novel Parallel Computing Systems

As discussed in Chapter 2, the future of the micromagnetic simulations may migrate to cloud computing. The benefits of utilizing multiple computing nodes in the cloud computing come in three folds: (1) it allows ultra-large scales simulations that does not fit into a single computing node; (2) it can accelerate the simulations by the capability of using a number of computing nodes; (3) it saves the energy throughput via the efficient management of the computational power in the “clouds”.

Another promising parallel computing technology is based on embedded systems, which come with a low energy consumption and low manufacturing cost. During the past few years, the embedded system became dominating the electronics consumption in the world and pushing the hardware development into a new stage. Modern embedded systems usually comprise multi-core CPU and GPU. The algorithms in the traditional desktop GPUs and multi-core CPUs can be applied to the embedded system without major changes in the implementation. For relatively small micromagnetic simulations, the use of embedded systems is especially promising. This creates opportunities to save the computational cost via the subdivision of the computational demands, which is important in the supercomputer centers. With the continuing increase of the computational power of

the embedded systems, there is a bright future to see high energy-and-cost efficient simulations on such systems.

6.2.2. Full GPU Implementation of FastMag

FastMag is capable of fully running on a single core or multiple cores of a multicore CPU, or partially offloading to GPU. Most of the computing-time bottlenecks are already offloaded to GPU to achieve a high speed. Provided with these optimization methods being applied, a full GPU version of FastMag could push the speed performance of FastMag still higher. The current CPU components may seem trivial, but in certain test-cases the computational time spent on operations, such as pointwise array operations, could not be ignored. These operations can be highly optimized if implemented on GPUs.

One concern of the full GPU implementation of a large-scale solver, such as FastMag, is that the GPU memory consumption could be too large. As a result, the problem size that the solver can handle may be limited by the GPU memory size. However, with the development of the GPU hardware technology, the memory capacity in the GPUs is also increasing rapidly. One of the latest GPGPU published by Nvidia, GTX Titan X, comes with 12 GB of memory in a single GPU, which is 6x greater than the relatively old GPU that is used in most of the numerical tests in here. It is reasonable to predict that the size of the GPU memory could go even larger in the future.

Another concern could be the difficulties in implementation since the complexity of the CPU code is already considerable. A possible solution could be OpenACC powered by Nvidia. It uses pragma statements before the targeting parallelization blocks to automatically parallelize the code that previously runs on CPU. This is not much more

difficult than the OpenMP implementation for multicore CPU. There have been a few successful projects using OpenACC on GPUs [108][109]. Therefore, it is worthwhile to investigate and try to integrate it into FastMag.

Bibliography

[1] Y. Nakatani, Y. Uesaka, N. Hayashi. "*Direct solution of the Landau-Lifshitz-Gilbert equation for micromagnetics*." Japanese Journal of Applied Physics, vol. 28, no. 12R, pp. 2485, 1989.

[2] S. Fu, L. Xu, V. Lomakin, A. Torabi, B. Lengsfield. "*Modeling perpendicular magnetic multilayered oxide media with discretized magnetic layers*." IEEE Transactions on Magnetics, vol. 51, no. 11, pp. 1-4, 2015..

[3] M. Lubarda, "*Micromagnetic Modeling and Analysis for Memory and Processing Applications*", Dissertation, UC San Diego, 2012.

[4] W.F. Brown, "*Domains, micromagnetics, and beyond: Reminiscences and assessments*." Journal of Applied Physics, vol. 49, no. 3, pp. 1937-1942, 1978.

[5] O. Chubykalo-Fesenko, U. Nowak, R.W. Chantrell, D. Garanin. "*Dynamic approach for micromagnetics close to the Curie temperature."* Physical Review B, vol. 74, no. 9, pp. 094436, 2006.

[6] S. Fu, R. Chang, S. Couture, M. Menarini, M. A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay, V. Lomakin, "*Micromagnetics on high-performance workstation and mobile computational platforms*," Journal of Applied Physics, vol. 117, no. 17, pp. 17E517, 2015.

[7] L. D. Landau, and E. M. Lifshitz, "*On the theory of the dispersion of magnetic permeability in ferromagnetic bodies*," Phys. Z. Sowietunion, vol. 8, no. 153, 1935.

[8] T.L. Gilbert, "*A Lagrangian formulation of the gyromagnetic equation of the magnetization field*.", Physical Review, vol. 100, pp. 1243, 1955.

[9] muMAG Micromagnetic Modeling Activity Group, http://www.ctcms.nist.gov/~rdm/mumag.org.html.

[10] B.D. Cullity, C.D. Graham. "*Introduction to magnetic materials*". John Wiley & Sons, 2011.

[11] Y. Luo, C. Zhou, C. Won, Y. Wu. "*Effect of Dzyaloshinskii–Moriya interaction on magnetic vortex*." AIP Advances, vol. 4, no. 4, pp. 047136, 2014.

[12] A. Fert, V. Cros, J. Sampaio. "*Skyrmions on the track*." Nature nanotechnology, vol. 8, no. 3 pp. 152-156., 2013.

[13] N. Kanazawa, J-H. Kim, D. Inosov, J. White, N. Egetenmeyer, J. Gavilano, S. Ishiwata et al. "*Possible skyrmion-lattice ground state in the B 20 chiral-lattice magnet*

MnGe as seen via small-angle neutron scattering." Physical Review B, vol. 86, no. 13, pp. 134425, 2012.

[14] M. Lee, W. Kang, Y. Onose, Y. Tokura, N. Ong, "*Unusual Hall effect anomaly in MnSi under pressure*." Physical review letters, vol. 102, no. 18, pp. 186601, 2009.

[15] A. Neubauer, C. Pfleiderer, B. Binz, A. Rosch, R. Ritz, P. Niklowitz, P. Böni. "*Topological Hall effect in the A phase of MnSi*." Physical review letters, vol. 102, no. 18, 186602, 2009.

[16] R. Chang, S. Li, M.V. Lubarda, B. Livshitz, V. Lomakin, "FastMag: Fast micromagnetic simulator for complex magnetic structures", Journal of Applied Physics, vol 109, no. 7, pp. 07D358, 2011.

[17] FEMME, A Multiscale Micromagnetic Finite Element Package, 2007.

[18] A. Kakay, E. Westphal, and R. Hertel, "*Speedup of FEM micromagnetic simulations with graphical processing units*," IEEE Transactions on Magnetics, vol. 46, issue. 6, pp. 2303, 2010.

[19] C. Abert, L. Exl, F. Bruckner, A. Drews, D. Suess, "*Magnum.fe: A micromagnetic finite-element simulation code based on FEniCS*", Journal of Magnetism and Magnetic Materials, vol. 345, pp. 29–35, 2013.

[20] M.J. Donahue and D.G. Porter, OOMMF User's Guide, Version 1.0, Interagency Report NISTIR 6376, National Institute of Standards and Technology, Gaithersburg, MD, 1999.

[21] A. Vansteenkiste, J. Leliaert, M. Dvornik, M. Helsen, F. Garcia-Sanchez, B. Van Waeyenberge, "*The design and verification of MuMax3*", AIP Advances, vol. 4, pp. 107133, 2014.

[22] "MicroMagnum." http://micromagnum.informatik.uni-hamburg.de/.

[23] R. Courant, Friedrichs, H. K. Lewy, "*Uber die partiellen di erenzengleichungen der mathematischen physik*," Mathematische Annalen, vol. 100, no. 1, pp. 32-74, 1928.

[24] R. Chang, "*Finite element and integral equation formulations for high -performance micromagnetic and electromagnetic solvers*", Dissertation. UC San Diego, 2014.

[25] J. Fidler, T. Schrefl, "*Micromagnetic modelling - the current state of the art*", Journal of Physics D: Applied Physics, vol. 33, no. 15, pp. R135, 2000.

[26] S.D. Cohen, A.C. Hindmarsh. "*CVODE, a stiff/nonstiff ODE solver in C*." Computational Physics. vol. 10, issue 2, pp. 138-143, 1996.

[27] L. Nyland, M. Harris, J. Prins, "*Fast n-body simulation with cuda*." GPU Gems 3, pp. 677, 2007.

[28] E. Süli, D. Mayers. "*An introduction to numerical analysis*". Cambridge University Press, 2003.

[29] T.Sato, Y. Nakatani. "*Fast micromagnetic simulation of vortex core motion by GPU*." Journal of the Magnetics Society of Japan, vol. 35, no. 3, pp. 163-170, 2011.

[30] D. Blythe, "*Rise of the graphics processor*," Proceedings of the IEEE, vol. 96, no.5, pp. 761-778, 2008.

[31] J. Bédorf, E. Gaburov, S.P. Zwart. "*A sparse octree gravitational N-body code that runs entirely on the GPU processor*." Journal of Computational Physics, vol. 231, no. 7, pp. 2825-2839, 2012.

[32] S. Fu, W. Cui, M. Hu, R. Chang, M.J. Donahue, V. Lomakin, "*Finite Difference Micromagnetic Solvers with Object Oriented Micromagnetic framework (OOMMF) on Graphics Processing Units*," IEEE Transactions on Magnetics, vol. 52, no. 4, pp. 1-9, 2016.

[33] M. Harris, "*CUDA Pro Tip: Do The Kepler Shuffle*", Parallel ForAll. https://devblogs.nvidia.com/parallelforall/cuda-pro-tip-kepler-shuffle, 2014.

[34] S.J. Pennycook, C.J. Hughes, M. Smelyanskiy, S.A. Jarvis. "*Exploring simd for molecular dynamics, using intel® xeon® processors and intel® xeon phi coprocessors*." Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 1085-1097. 2013.

[35] K. Rupp, "*CPU, GPU and MIC hardware characteristics over time.*" Online blog, https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/, 2013.

[36] D. Kondo, B. Javadi, P. Malecot, F. Cappello, D. Anderson. "*Cost-benefit analysis of cloud computing versus desktop grids*." Parallel & Distributed Processing 2009, IEEE International Symposium on, pp. 1-12. 2009.

[37] D. Hassabis, "*Official Google Blog: What we learned in Seoul with AlphaGo*.", https://googleblog.blogspot.com/2016/03/what-we-learned-in-seoul-with-alphago.html, 2016.

[38] R. Coulom, "*Computing elo ratings of move patterns in the game of go*." Computer Games Workshop, 2007.

[39] "Nvidia, OEM Partners Show How GPUs Are Used in Data Center [News]", eWeek, http://www.eweek.com/servers/slideshows/nvidia-oem-partners-show-how-gpus-are-used-in-data-center.html, 2015.

[40] "*Tesla gpu accelerators for servers*", http://www.nvidia.in/object/tesla-server-gpus-in.html, 2016.

[41] C. Jermain 2016, G. Rowlands, R. Buhrman, D. Ralph, "*GPU-accelerated micromagnetic simulations using cloud computing*.", Journal of Magnetism and Magnetic Materials, vol. 401, pp. 320-322, 2016.

[42] "GPU GFlops", 2014, URL: http://kyokojap.myweb.hinet.net/gpu_gflops/.

[43] C. Abert, F. Bruckner, C. Vogler, R. Windl, R. Thanhoffer, D. Suess. "*A full-fledged micromagnetic code in fewer than 70 lines of NumPy*." Journal of Magnetism and Magnetic Materials, vol. 387, pp. 13-18, 2015.

[44] L. Lopez-Diaz, D. Aurelio, L. Torres, E. Martinez, M. A. Hernandez-Lopez, J. Gomez, O. Alejos, M. Carpentieri, G. Finocchio, and G. Consolo. "*Micromagnetic simulations using graphics processing units*." Journal of Physics D: Applied Physics, vol. 45, no. 32 pp. 323001, 2012.

[45] C. Abert, G. Wautischer, F. Bruckner, A. Satz, D. Suess. "*Efficient energy minimization in finite-difference micromagnetics: Speeding up hysteresis computations*." Journal of Applied Physics, vol. 116, no. 12, pp. 123908 , 2014.

[46] T. Schrefl, J. Fidler, K. J. Kirk and J. N. Chapman, "*A higher order FEM-BEM method for the calculation of domain processes in magnetic nano-elements*," IEEE Transactions on Magnetics, vol. 33, no. 5, pp. 4182-4184, 1997.

[47] L. Greengard, R. Vladimir. "A fast algorithm for particle simulations." Journal of computational physics, vol. 73, issue 2, pp. 325-348, 1987.

[48] A. Boag, B. Livshitz. "Adaptive nonuniform-grid (NG) algorithm for fast capacitance extraction." Microwave Theory and Techniques, IEEE Transactions on, Issue 54, vol. 9, pp. 3565-3570, 2006.

[49] K. Fabian, A. Kirchner, W. Williams, F. Heider, T. Leibl, A. Huber. "Three dimensional micromagnetic calculations for magnetite using FFT." Geophysical Journal International, vol. 124, issue 1, pp. 89-104, 1996.

[50] M. Frigo and S.G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE, vol. 93, Issue 2, pp. 216–231, 2005.

[51] Intel Corp. http://software.intel.com/en-us/intel-mkl/.

[52] CUDA CUFFT Library, Version 7.5. NVIDIA Corp., 2013.

[53] S. Li, B. Livshitz, V. Lomakin, "Graphics Processing Unit Accelerated O(N) Micromagnetic Solver", Magnetics, IEEE Transactions on, vol. 46, no. 6, pp. 2373 – 2375, 2010.

[54] Z. Li, S. Zhang. "Magnetization dynamics with a spin-transfer torque." Physical Review B, vol. 68, issue. 2, pp. 024404, 2003.

[55] A.J. Newell, J. Andrew, W. Williams, and D.J. David, "A generalization of the demagnetizing tensor for nonuniform magnetization." Journal of Geophysical Research, Solid Earth (1978–2012) vol. 98, no. B6, pp. 9551-9555, 1993.

[56] B. Van de Wiele , F. Olyslager, L. Dupre, D. De Zutter, "On the accuracy of FFT based magnetostatic field evaluation schemes in micromagnetic hysteresis modeling," Journal of Magnetism and Magnetic Materials, vol. 322, no. 4, pp. 469-476, 2010.

[57] R.D. McMichael, M.J. Donahue, D.G. Porter, J. Eicke, "Comparison of magnetostatic field calculation methods on two-dimensional square grids as applied to a micromagnetic standard problem," Journal of Applied Physics, vol.85, no.8, pp. 5816-5818, 1999.

[58] C. Abert, G. Selke, B. Kruger, A. Drews, "*A Fast Finite-Difference Method for Micromagnetics Using the Magnetic Scalar Potential*," IEEE Transactions on Magnetics, vol. 48, Issue 3, pp. 1105–1109, 2012.

[59] M.J. Donahue, "Parallelizing a micromagnetic program for use on multi-processor shared memory computers." IEEE Transactions on Magnetics, vol. 45, Issue 10, pp. 3923-3925, 2009.

[60] "The Object Oriented MicroMagnetic Framework (OOMMF) project at ITL/NIST", http://math.nist.gov/oommf/.

[61] "OOMMF on GPU", http://cem.ucsd.edu/OOMMF.html

[62] M. Harris, "Optimizing parallel reduction in CUDA", Nvidia Developer Technology 2, Nvidia Corp., 2007.

[63] J.H. Byun, R. Lin, K. A. Yelick, J. Demmel, "Autotuning Sparse Matrix-Vector Multiplication for Multicore", EECS Department, University of California, Berkeley, (2012). http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-215.html.

[64] E. J. Im, K. Yelick, and R. Vuduc. "Sparsity: Optimization framework for sparse matrix kernels." International Journal of High Performance Computing Applications, vol. 18, no. 1, pp. 135-158, 2004.

[65] J. R. Gilbert, E. G. Ng, B. W. Peyton. "An efficient algorithm to compute row and column counts for Sparse Cholesky Factorization." SIAM Journal on Matrix Analysis and Applications, vol. 15, no. 4, pp. 1075-1091, 1994.

[66] P. R. Amestoy, T. A. Davis, I. S. Duff. "An approximate minimum degree ordering algorithm." SIAM Journal on Matrix Analysis and Applications, vol. 17, no. 4, pp. 886-905, 1996.

[67] W. H. Liu, A. H. Sherman. "Comparative analysis of the Cuthill-Mckee and the Reverse Cuthill-Mckee ordering algorithms for sparse matrices." SIAM Journal on Numerical Analysis 13, no. 2, pp. 198-213, 1976.

[68] I. P. King. "An automatic reordering scheme for simultaneous equations derived from network systems." International Journal for Numerical Methods in Engineering, vol. 2, no. 4, pp. 523-533, 1970.

[69] A. Pinar, M. T. Heath. "Improving performance of sparse matrix-vector multiplication." Proceedings of the 1999 ACM/IEEE conference on Supercomputing, ACM, pp. 30, 1999.

[70] "NVLink, Pascal and Stacked Memory: feeding the appetite for big data." Nvidia. http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/

[71] N. Bell, M. Garland. "Implementing sparse matrix-vector multiplication on throughput-oriented processors." Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, pp. 18, 2009.

[72] X. Yang, S. Parthasarathy, P. Sadayappan. "Fast sparse matrix-vector multiplication on GPUs: Implications for graph mining." Proceedings of the VLDB Endowment, vol. 4, no. 4, pp. 231-242, 2011.

[73] R.H. Victora, "Quantitative theory for hysteretic phenomena in CoNi magnetic thin films", Physical Review Letters, vol. 58, pp. 1788-1791, 1987.

[74] M. Mansuripur, "Magnetization reversal dynamics in the media of magneto-optical recording", Journal of Applied Physics, vol. 63, pp. 5809-5823, 1988.

[75] J. Zhu, H.N.l Bertram "*Magnetization reversal in CoCr perpendicular thin films*", Journal of Applied Physics. vol. 66, no. 3, pp. 1291-1307, 1989.

[76] K.M. Tako, M.A. Wongsam, R.W. Chantrell, "Numerical simulation of 2D thin films using a finite element method", Journal of Magnetism and Magnetic Materials, vol. 155, no. 1-3, pp. 40-42, 1996.

[77] D. Suess, V. Tsiantos, T. Schrefl, J. Fidler, W. Scholz, H. Forster, R. Dittrich, J.J. Miles, "Time resolved micromagnetics using a preconditioned time integration method". Journal of Magnetism and Magnetic Materials, vol. 248, no. 2, pp. 298-311, 2002.

[78] D. Shepherd, J.J. Miles, M. Heil, M. Mihajlovic, "Discretization-induced stiffness in micromagnetic simulations", Magnetics, IEEE Transactions on, vol. 50, no. 11, pp. 1-4, 2014.

[79] K.E. Atkinson, "*An introduction to numerical analysis*". John Wiley & Sons, 2008.

[80] E. Fehlberg, "*Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems*." NASA Technical Report 315, 1969.

[81] E. Hairer, S. Nrsett, G. Wanner. "*Solving Ordinary Differential Equations: Nonstiff problems. v. 2: Stiff and differential-algebraic problems*". Springer Verlag, 2010.

[82] V. Tsiantos, J.J. Miles. "*Fast micromagnetic simulations using an analytic mathematical model*." Physica B: Condensed Matter, vol. 372, no. 1, pp. 303-307, 2006.

[83] J. D. Lambert. "*Computational methods in ordinary differential equations*". John Wiley & Sons, London-New York-Sydney, 1973.

[84] V. Tsiantos, J.J. Miles, M. Jones, "*Preconditioned krylov subspace methods in micromagnetic simulations*". European Congress on Computational Methods in Applied Sciences and Engineering, 2000.

[85] C. Pommerell, "*Solution of large unsymmetric systems of linear equations*", PhD thesis, ETH, 1992.

[86] Y. Saad, "Iterative Methods for Sparse Linear Systems", 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[87] R. Wood. "The feasibility of magnetic recording at 1 terabit per square inch." Magnetics, IEEE Transactions on, vol. 36, no. 1 pp. 36-42, 2000.

[88] Tom Coughlin, "*Seagate Pushes HDD Areal Density Growth To About 60% For 2015*", http://www.forbes.com/sites/tomcoughlin/2015/09/03/seagate-pushes-hdd-areal-density-growth-to-about-60-for-2015/#65030dba588e, Forbes, 2015.

[89] H. Takano, Y. Nishida, A. Kuroda, H. Sawaguchi, Y. Hosoe, T. Kawabe, H. Aoi, H. Muraoka, Y. Nakamura, K. Ouchi. "*Realization of 52.5 Gb/in 2 perpendicular recording.*" Journal of magnetism and magnetic materials, vol. 235, no. 1, pp 241-244, 2001.

[90] Garth Gibson, Milo Polte, "*Directions for Shingled-Write and Two-Dimensional Magnetic Recording System Architectures: Synergies with Solid-State Disks*", Carnegie Mellon University Parallel Data Lab Technical Report, CMU-PDL-09-104, 2009.

[91] Xiaobin Wang, H. Neal Bertram, and Vladimir L. Safonov, "*Thermal-Dynamic Reversal Of Fine Magnetic Grains With Arbitrary Anisotropy Axes Orientation*", Journal of Applied Physics, vol. 92, no. 4, pp. 2064-2072, 2002.

[92] R. Rottmeyer, S. Batra, D. Buechel, W. Challener, J. Hohlfeld, Y. Kubota, L. Li et al., "*Heat-Assisted Magnetic Recording*," Magnetics, IEEE Transactions on, vol. 42, no. 10, pp. 2417–2421, 2006.

[93] JG Zhu, X Zhu, Y Tang, "*Microwave Assisted Magnetic Recording*," Magnetics, IEEE Transactions on, vol. 44, no. 1, pp. 125-131 , 2008.

[94] R. Wood, M. Williams, A. Kavcic, J. Miles. "*The Feasibility Of Magnetic Recording At 10 Terabits Per Square Inch On Conventional Media*." Magnetics, IEEE Transactions on, vol. 45, no. 2 pp. 917-923, 2009.

[95] B. Terris, T. Thomson, G. Hu, "*Patterned media for future magnetic data storage*," Microsystem Technologies, vol. 13, no. 2, pp. 189–196, 2006.

[96] R. Wood, R. Galbraith, J. Coker. "*2-D magnetic recording: Progress and evolution*." Magnetics, IEEE Transactions on, vol. 51, no. 4, pp. 1-7, 2015.

[97] "ASTC Technology Roadmap - 2014 v8", available at http://idema.org/wp-content/plugins/download-monitor/download.php?id=2244.

[98] Iwasaki, Shun-ichi. "*Perpendicular magnetic recording*." Magnetics, IEEE Transactions on, vol. 16, no. 1, pp. 71-76, 1980.

[99] Y. Nakamura, I. Tagawa, "*Possibilities of perpendicular magnetic recording*", Magnetics, IEEE Transactions on, vol. 24, no. 6, pp. 2329-2334, 1988.

[100] "Perpendicular-eng.jpg", https://commons.wikimedia.org/wiki/File:Perpendicular-eng.jpg

[101] E.E. Fullerton, J. S. Jiang, M. Grimsditch, C. H. Sowers, S. D. Bader. "*Exchange-spring behavior in epitaxial hard/soft magnetic bilayers*." Physical Review B, vol. 58, no. 18, pp. 12193, 1998.

[102] R. H. Victora and X. Shen, "Composite Media for Perpendicular Magnetic Recording," Magnetics, IEEE Transactions on, vol. 41, no. 2, pp. 537-542, 2005.

[103] G. Choe, J. Park, Y. Ikeda, B. Lengsfield, T. Olsen, K. Zhang, S. Florez and A. Ghaderi, "Writeability enhancement in perpendicular magnetic multilayered oxide media for high areal density recording," Magnetics, IEEE Transactions on, vol. 47, no. 1, pp. 55-62, 2011.

[104] M. P. Wismayer, S.L. Lee, T. Thomson, F.Y. Ogrin, C.D. Dewhurst, S. M. Weekes and R. Cubitt, "Using small-angle neutron scattering to probe the local magnetic structure of perpendicular magnetic recording media," Journal of applied physics, vol. 99, no. 8, p. 08E707, 2006.

[105] L. Saharan, C. Morrison, J. J. Miles, T. Thomson, T. Schrefl and G. Hrkac, "Angle dependence of the switching field of recording media at finite temperatures," Journal of applied physics, vol. 110, no. 10, p. 103906, 2011.

[106] J. Zhu, T. Lam, Y. Luo and X. Ye, "Nonlinear partial erasure and its correlation with transition noise in longitudinal thin-film media," Journal of applied physics, vol. 79, no. 8, pp. 4906-4908, 1996.

[107] K. Tang, K. Takano, G. Choe, G. Wang, J. Zhang, X. Bian and M. Mirzamaani, "A Study of Perpendicular Magnetic Recording Media With an Exchange Control Layer," Magnetics, IEEE Transactions on, vol. 44, no. 11, pp. 3507-3510, 2008.

[108] S. Feki, A. Al-Jarro, H. Bagci. "*Multi-GPU-based acceleration of the explicit time domain volume integral equation solver using MPI-OpenACC*." In Radio Science Meeting (Joint with AP-S Symposium), 2013 USNC-URSI, pp. 90-90. IEEE, 2013.

[109] M. Otten, J. Gong, A. Mametjanov, A. Vose, J. Levesque, P. Fischer, M. Min. "*An MPI/OpenACC implementation of a high-order electromagnetics solver with GPUDirect communication*." International Journal of High Performance Computing Applications, 2016.