Finite Difference Micromagnetic Solvers with the Object Oriented MicroMagnetic Framework (OOMMF) on Graphics Processing Units

Sidi Fu[1,2], Weilong Cui[3], Matthew Hu[2], Ruinan Chang[1,2], Michael J. Donahue[4], and Vitaliy Lomakin[1,2]

[1]Center for Magnetic Research, University of California, San Diego, CA 92037 USA
[2]Department of Electrical and Computer Engineering, University of California, San Diego, CA 92037 USA
[3]Department of Electronic Engineering and Computer Science, Peking University, Beijing, 100871 CHINA
[4]National Institute of Standards and Technology, Gaithersburg, MD 20899-8910 USA

A micromagnetic solver using the Finite Difference method on a Graphics Processing Unit (GPU) and its integration with the Object Oriented MicroMagnetic Framework (OOMMF) are presented. Two approaches for computing the magnetostatic field accelerated by the Fast Fourier Transform (FFT) are implemented. The first approach, referred to as the tensor approach, is based on the tensor spatial convolution to directly compute the magnetostatic field from magnetic moments. The second approach, referred to as the scalar potential approach, uses differential operator evaluation through finite differences (divergence for magnetic charge and gradient for magnetostatic field) and spatial convolution for magnetic scalar potential. Comparisons of implementation details, speed, memory consumption and accuracy are provided. The GPU implementation of OOMMF shows up to 32x GPU-CPU speed-up.

***Index Terms*—Micromagnetics, Graphics Processing Unit, OOMMF, magnetostatics.**

I. Introduction

MICROMAGNETIC solvers for the Landau-Lifshitz-Gilbert (LLG) equation are important for our ability to analyze and design many magnetic systems. The two main types of micromagnetic solvers are based on the Finite Element method (FEM) [1]-[4] and Finite Difference method (FDM) [5]-[7]. FEM is flexible for various geometries, while FDMs often have higher performance for relatively simple structures, such as thin films. In FDMs, the exchange field is typically evaluated via the Laplacian operator using finite differences. The magnetostatic field can be evaluated through several approaches such as the fast multipole method [8], the non-uniform grid interpolation method [9], and the Fast Fourier Transform (FFT) [10]. The last approach is most commonly used for FDMs. The FFT reduces the computation complexity from $O(N^2)$ to $O(N\log N)$. With well-developed numerical libraries [11]-[13], FFTs can be applied to tensor or scalar potential formulations with associated advantages for each approach.

The speed limitation of single-core computer systems has become an obstacle when solving large-scale problems in micromagnetics and other fields of study. Micromagnetic solvers for multicore and multi-CPU systems have been developed, but such systems have limitations in their performance. Massively parallel Graphics Processing Unit (GPU) 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 [7], [14]-[16].

This paper introduces a scalar potential FDM approach using only one (forward and inverse) scalar FFT and compares it to a more conventional tensor FDM method. The computational domain is discretized into a uniform grid of identical rectangular brick cells and the magnetization is set inside each cell. The magnetostatic field evaluation is usually the most time and memory consuming part of the solver. The tensor approach, most commonly used in the micromagnetics community, involves superposition integrals with a tensor integral kernel to directly compute the magnetostatic field. The scalar approach introduced here involves evaluating the scalar potential via superposition integrals with a scalar integral kernel, accompanied with differential operators evaluated via finite differences. This approach is related to that of [17]-[19], but it uses scalar charges explicitly, which is critical to minimize the memory consumption and the number of FFTs, and it is applicable to 3D structures. The tensor and scalar potential approaches are compared in terms of their formulation, implementation on GPUs, accuracy, and performance results. Approaches for efficiently computing the exchange field are presented. The paper, then, demonstrates a GPU implementation of the widely used Object Oriented Micromagnetic Framework (OOMMF) [5]. 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 greater speed. The GPU-accelerated OOMMF will be made freely available to the micromagnetic community at the OOMMF web site [20].

II. Formulation

A. Landau-Lifshitz-Gilbert equation

This paper is concerned with solving the explicit form of LLG equation via FDM. In FDMs, the structure of interest is discretized into identical brick cells (i.e., rectangular prisms). Each cell can have different material parameters and is assigned a magnetization state. The discretized LLG equation reads

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

where $\mathbf{m}_i$ is the normalized magnetization at the center of the corresponding cell i, γ is the gyromagnetic ratio, α is the Gilbert damping constant, and $\mathbf{H}_{eff}$ is the effective field. The effective field is comprised of the applied (or Zeeman) field $\mathbf{H}_{appl}$, the anisotropy field $\mathbf{H}_{anis}$, the exchange field $\mathbf{H}_{exch}$, and the self-magnetostatic field $\mathbf{H}_{ms}$. Additional effective field and torque terms in the LLG equation may be added, e.g. spin transfer torques [21].

B. Self-Magnetostatic Field

The self-magnetostatic field can be defined as a superposition integral

$$\mathbf{H}_{\mathrm{ms}}(r) = M_s \nabla\nabla \iiint_{V'} \frac{\mathbf{m}(\mathbf{r}')}{4\pi|\mathbf{r}-\mathbf{r}'|} dV' . \qquad (2)$$

Two approaches are implemented to compute the magnetostatic field: the tensor approach and the scalar potential approach. The tensor approach solves for the magnetostatic field directly using superpositions with the tensor integral kernel. The scalar potential approach uses scalar charges to find the scalar potential, which is used to compute the field. The tensor approach is most widely used in micromagnetics, but the scalar potential approach introduced here has lower computational cost and memory requirements.

1) Tensor Approach

In the tensor formulation the double del operator is moved under the integral to give

$$\mathbf{H}_{\mathrm{ms}}(\mathbf{r}) = M_s \iiint_{V'} \left(\nabla\nabla \frac{1}{4\pi|\mathbf{r}-\mathbf{r}'|} \right) \cdot \mathbf{m}(\mathbf{r}') dV' . \qquad (3)$$

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

$$\mathbf{H}_{\mathrm{ms},i} = M_s \sum_{j=1}^{N} \mathbf{N}_{ij} \mathbf{m}_j ; \;\; \mathbf{N}_{ij} = V_i^{-1} \int_{S_i} \int_{S_j} \frac{d\mathbf{S}_i d\mathbf{S}_j}{4\pi|\mathbf{r}_i - \mathbf{r}_j|} , \qquad (4)$$

where V_i is the volume of the of cell i, and $d\mathbf{S}_i$ and $d\mathbf{S}_j$ represent surface integrals over the surfaces of the cells i and j, respectively. The tensor $\mathbf{N}_{ij}$ provides the field generated by the magnetization in cell j at cell i and can be computed as outlined in [22].

The direct cost of computing the magnetostatic field across the entire simulation via Eq. (4) is $O(N^2)$. However, using the fact that the discretization is uniform, the summation in Eq. (4) can be computed via a three-dimensional Fast Fourier Transform (3D FFT), which reduces the computational cost to $O(N \log N)$.

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

The number of operations in the tensor approach per time step is $3c_1 N \log N$, where the constant c_1 is related to a single scalar 3D FFT evaluation. The memory storage is $30N$ real numbers. The rate of numerical convergence for decreasing cell size D is of $O(D^{-2})$ [23], [24].

2) Scalar Potential Approach

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:

$$\begin{aligned} &\rho_M = -M_s \nabla \cdot \mathbf{m}; \; \sigma_M = M_s \hat{\mathbf{n}} \cdot \mathbf{m} \\ &\phi(\mathbf{r}) = \iiint_V \frac{\rho_M(\mathbf{r}')}{4\pi|\mathbf{r}-\mathbf{r}'|} dV' + \oiint_S \frac{\sigma_M(\mathbf{r}')}{4\pi|\mathbf{r}-\mathbf{r}'|} dS' . \qquad (5) \\ &\mathbf{H}_{\mathrm{ms}}(\mathbf{r}) = -\nabla\phi(\mathbf{r}) \end{aligned}$$

Here ρ_M and σ_M are volume and surface charge densities, respectively, and 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. If the magnetization locations are assigned at the discretization cell centers, the surface charge locations are shifted by a half-cell with respect to the volume charges. One simple approach is to consider the volume and surface charges and their corresponding potential parts separately [25]. However, while allowing for reducing the error such an approach would require evaluating the integrals for the potential twice. The approach we use is to find the magnetization states at the cell vertices by averaging over the magnetization states defined in the centers of the cells surrounding the vertices, i.e.,

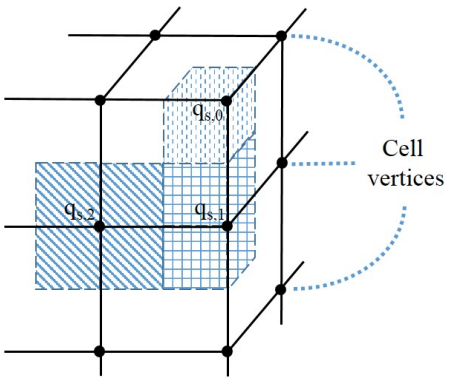

Fig. 1. Retrieval of the surface charges at the vertices of cells by averaging over the surrounding surfaces. Three kinds of surface charges ($q_{s,0}$, $q_{s,1}$, $q_{s,2}$) and the corresponding surface areas are illustrated.

$$\mathbf{m}_{i_n} = \sum_{i_c} \mathbf{m}_{i_c(i_n)} \Big/ I_c(i_n) , \qquad (6)$$

where for each vertex i_n the summation is over the cells $i_c(i_n)$ surrounding this vertex and $I_c(i_n)$ is the total number of surrounding cells. The vertex magnetization values are used to find the surface charges at the vertices at the boundaries. Specifically, the surface charges are found as a sum of the surface charges on the surfaces surrounding the vertices (Fig. 1):

$$q_{s,i_n} = \sum_{i_s(i_n)} (M_{s,i_s(i_n)} \mathbf{m}_{i_n} \cdot \hat{\mathbf{n}}_{i_s(i_n)}) s_{i_s(i_n)} , \qquad (7)$$

where i_n is the boundary vertex numbers, $\hat{\mathbf{n}}_{i_s(i_n)}$ is the normal corresponding to the surrounding boundary surface, 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 accounted for more than once per surface vertex, e.g., if the structure is discretized into brick cells with the side surface area of Δs , then $s_{i_s(i_n)} = \Delta s/4$. The volumetric charges q_{v,i_n} corresponding to the vertices are all found by finite differences. In this approach the volumetric and surface charges are collocated at the vertices, which allows lumping them together for the purpose of computing the scalar potential:

$$\phi_{i_n} = \sum_{\substack{j_n=1 \\ \mathbf{r}_{i_n} \neq \mathbf{r}_{j_n}}}^{N_n} \frac{q_{j_n}}{4\pi |\mathbf{r}_{i_n} - \mathbf{r}_{j_n}|} + \phi_{i_n}^{self} \ , \tag{8}$$

where $q_{j_n} = q_{v,j_n}$ at the interior vertices and $q_{j_n} = q_{v,j_n} + q_{s,j_n}$ at the boundary vertices. 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 vertex. The computation can be made more accurate by considering that the charges q_{j_n} are spread into volumetric charge densities in cells centered at the vertex and replacing the factor $|\mathbf{r}_{i_n} - \mathbf{r}_{j_n}|^{-1}$ with the integral $\int_{V_{j_n}} (q_{j_n}/\Delta v_{j_n}) |\mathbf{r}_{i_n} - \mathbf{r}_{j_n}|^{-1} dv$. This method is not employed in the numerical tests since it is not the bottleneck of the accuracy for the scalar potential approach. The summation in Eq. (8) or the integrals in Eq. (5) can be treated as three dimension scalar convolutions via FFT. Once the potentials at the vertices are found, the magnetostatic field is found by taking finite differences at the centers of the discretization cells.

As compared to the tensor approach, the scalar potential approach has reduced computational cost and memory consumption. In particular, the number of FFTs is reduced from 6 to 2, and the memory consumption is reduced by about one third. The scalar potential approach here also shows similar advantages over the scalar potential approach in [17]-[19]: the number of FFTs is reduced from 4 to 2, and the memory consumption is reduced to about one half. The overall accuracy of the scalar potential approach scales quadratically with respect to the discretization cell size, similar to the tensor approach, although as shown in Sec. IV the tensor approach is typically more accurate by a constant factor for the same discretization cell size.

The variables in the scalar potential approach are defined on the cell vertices, leading to the problem size of $(n_x+1)(n_y+1)(n_z+1)$ as compared to $n_x n_y n_z$ in the tensor approach, where n denotes the number of cells. This leads to noticeable difference in small-scale or reduced dimensionality problems, such as magnetic thin films ($n_z = 1$). In the latter case the scalar potential can be calculated via four 2D FFTs, which is only slightly slower than the tensor method (with three 2D FFTs). In general, the scalar potential approach is appealing for solving large 3D problems.

C. Other Effective Field Components

The exchange field in the continuous form is given by

$$\mathbf{H}_{\text{exch}} = \frac{2A}{\mu_0 M_S} \nabla^2 \mathbf{m} \ , \tag{9}$$

where A is the exchange constant. We use the nearest neighbor finite difference rule, which for the interior cells reads as

$$\mathbf{H}_{\text{exch},i} = \frac{2A}{\mu_0 M_{S,i}} \sum_{j \in J_i} \frac{(\mathbf{m}_j - \mathbf{m}_i)}{h_{ij}^2} \ , \tag{10}$$

where J_i is a list of cells surrounding cell i , and h_{ij} is the distance between the center of cells. This rule is modified at boundary cells to maintain the boundary conditions on $\partial \mathbf{m}/\partial n$. The accuracy of the exchange field computation via Eq. (10) scales quadratically with respect to the discretization cell size, which is the same rate as the accuracy of the magnetostatic field [24]. The computation of the exchange field involves $O(N)$ operations. Its implementation on GPUs requires properly addressing the GPU memory and computations as outlined in Sec. III.

The uniaxial anisotropy field with the easy axis in the direction of a unit vector $\mathbf{k}$ is given by

$$\mathbf{H}_{anis,i} = \frac{2K_i}{\mu_0 M_{S,i}} (\mathbf{m}_i \cdot \mathbf{k}_i) \mathbf{k}_i \ , \tag{11}$$

where K is the magneto-crystalline anisotropy energy density, and M_s is the saturation magnetization. Other types of anisotropy fields, e.g., second order or cubic anisotropy, can be defined. The applied field can be simply assigned to each discretization cell as its value at the cell center. The applied and anisotropy fields involve only local operations, so their operation count is $O(N)$, and their numerical implementation is trivial.

III. CUDA Implementation

The GPU implementation of both tensor and scalar approaches is similar in that both methods require evaluating superpositions via FFTs and differential operators.

A. General Implementation Strategy

The GPU used in this work is Nvidia GeForce GTX 690, which has dual GPUs and each GPU has 1536 streaming processors. These processors are launched by CUDA threads in the CUDA programming environment [26]. A group of 32 threads executes the same instruction at a given time, which is called “atomic” behavior.

A certain GPU global memory is accessible to all threads. For example, a single GPU device of a dual-GPU Nvidia GeForce GTX 690 (used in this paper) has 2 GB global memory and the Titan X GPU has 12 GB of global memory. GPU global memory is separate from CPU memory, so any data that needs to be operated on by the GPU has to be transferred from the CPU memory. The GPU global memory has significant access latency. This latency is minimized by reducing the memory bandwidth. This can be achieved when reading a block of data from a continuous set of addresses in

the memory via so called "coalesced access".

The GPU also has a category of "shared memory". Shared memory is fast on-chip memory and it works as a memory pool for the threads to share intermediate data. Unlike global memory, shared memory has low intrinsic latency. However, access can be slowed by bank conflicts. In Nvidia GPUs, the shared memory is organized into memory banks. If there are two threads within one warp trying to access the same bank, these two accesses have to be serialized. Thus a linear accessing stride is preferred [26].

B. Evaluating the Superposition Summations

As discussed in Sec. II, 3D FFTs are used for accelerating the superposition summations in the magnetostatic field computation. We use the Nvidia cuFFT library. The 3D real-to-complex Fourier transform and its inverse complex-to-real Fourier transform are utilized to gain a good GPU-CPU speed-up while minimizing the GPU memory consumption.

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 [17]-[19] 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 N 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 [27]. 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.

C. Differential Operators

The divergence and gradient operators in the scalar potential approach for magnetostatic field computation, and the Laplacian operator in the computation of the exchange field are important differential operators. Provided that our implementation of the integral operators is sufficiently efficient, these differential operators may become the bottleneck if they are not efficiently implemented.

Differential operators are evaluated based on information from adjacent cells or vertices. Consider the Laplacian operator evaluation in Eq. (10). Since we use the 6-neighbor exchange field model to have second order numerical accuracy, the magnetization data in each cell will be read 6 times through CUDA threads launched for its neighboring cells. Even with coalesced memory access, the memory loading is still relatively heavy as compared to the computational workload. We take advantage of the shared memory on each streaming multiprocessor to avoid reading the magnetization data from global memory multiple times. Through fetching blocks of data from the global to shared memory, we perform all the following memory loads within the fast shared memory. Since the blocks of data overlap with each other by 1 layer of cells, data reuse is enhanced with larger block sizes. However, the block size is limited by the size of the shared memory in the GPU. Therefore, the optimized block size varies among different simulated geometries. Through the utilization of shared memory, the speed of the exchange field kernel on GPU is accelerated about 40%. A similar approach is used for computing the charges in the scalar potential method.

Another option to implement the exchange field is to include it in the computation of the magnetostatic field by adding it to the integral kernel. This is possible because the finite difference operator can be cast in a convolutional form. In this approach, the computation of the exchange field does not add any cost on top of the magnetostatic field computation [15]. However, we prefer to keep the exchange field as a separate effective field kernel for three reasons: (i) the exchange field cannot be integrated with the magnetostatic field in the scalar potential approach, (ii) a separate exchange field kernel provides flexibility for implementing implicit time evolution methods in future OOMMF updates, and (iii) computing the exchange field separately adds only a small fraction to the overall computation time (around 10%).

D. 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 on every time step, we have 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 [28] 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 Sec. IV.

E. Other Important Kernels

Evaluating the applied and anisotropy fields at the grid points does not require any information from other grid points. This workload can be fully parallelized through the "one thread per grid cell" approach. The computational complexity of the applied and anisotropy field evaluations is $O(N)$. To reduce the global memory accessing time, magnetization loading was carefully ordered so that the memory reading was always coalesced.

IV. Testing Results and Discussion

This section presents computational results of our implementation of the FDM solver with the scalar potential and tensor approaches, and also results from the OOMMF solver. First we validated our numerical results for both solvers with the µMag standard problem 4 [29] and by checking domain wall motion in a nanowire. Then, numerical accuracy and timing results were compared for the two methods of computing the magnetostatic field. Finally, we present the timing results from the OOMMF solver running on a CPU and a GPU. The CPU results were obtained on a 3.5GHz Intel Xeon E5 with 6 cores, while the GPU results were obtained on a Nvidia Geforce GTX 690 GPU. The CPU-GPU memory transfers in the micromagnetic solvers occur only before and after the simulation, and are therefore not included in the timing results. Unless specified, all the GPU results presented are from single-precision floating point operations with an explicit time-stepping method.

A. Validation

1) µMag Standard Problem 4

The µMag standard problem 4 is designed to simulate the dynamic magnetization process by considering the magnetization reversal in a thin film having dimensions 500 nm x 125 nm x 3 nm (see the µMag website [29], [30]). This problem includes exchange, self-magnetostatic, and applied fields, but the magneto-crystalline anisotropy is zero. The equilibrium S-state initial magnetization configuration is reversed by applying one of two fields; in Prob. 4a the applied field is $\mu_0\mathbf{H}_x = -24.6$ mT, $\mu_0\mathbf{H}_y = 4.3$ mT, $\mu_0\mathbf{H}_z = 0.0$ mT, while Prob. 4b uses $\mu_0\mathbf{H}_x = -35.5$mT, $\mu_0\mathbf{H}_y = -6.3$ mT, $\mu_0\mathbf{H}_z = 0.0$ mT. The discretization size for this test was chosen to be 2.5 nm x 2.5 nm x 3 nm. Single-precision floating point was used to obtain the best GPU performance. Both the tensor approach with adaptive Euler time evolver and scalar approaches were used.

As required by the standard problem, the (x, y, z) components of the spatially averaged magnetization as a function of time from S-state to the equilibrium (the first 1 ns) and the snapshot of magnetization when the averaged Mx reaches zero for the first time are shown in Fig. 2 and Fig. 3.

In Figs. 2 and 3 The results obtained via the tensor approach were indistinguishable visually with the reference case and, therefore, we refer to them as reference results. We can find that the results obtained via the scalar approach are in agreement with the reference results. Some of the differences are due to some numerical difference between the magnetostatic fields obtained via the tensor and scalar methods (see Sec. IV.B).

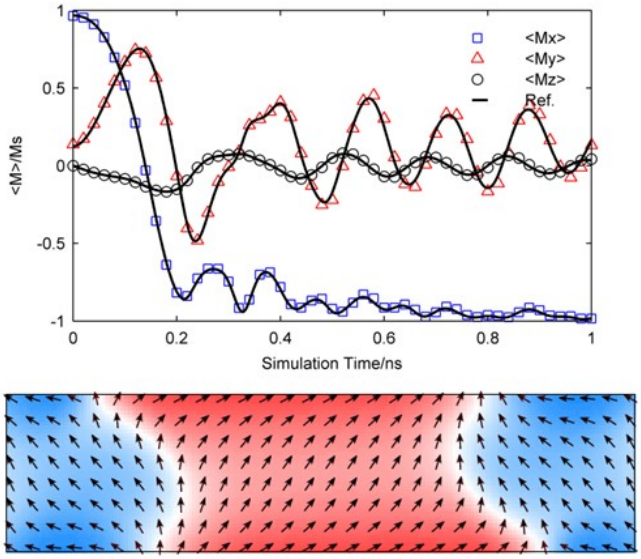

Fig. 2. The dynamics of averaged magnetization and a snapshot of the magnetization when averaged $\mathbf{M}_x$ first becomes zero, as specified by µMag standard problem 4a. In the top figure, the results from our solver are marked by colored symbols, while the reference results are solid black lines.

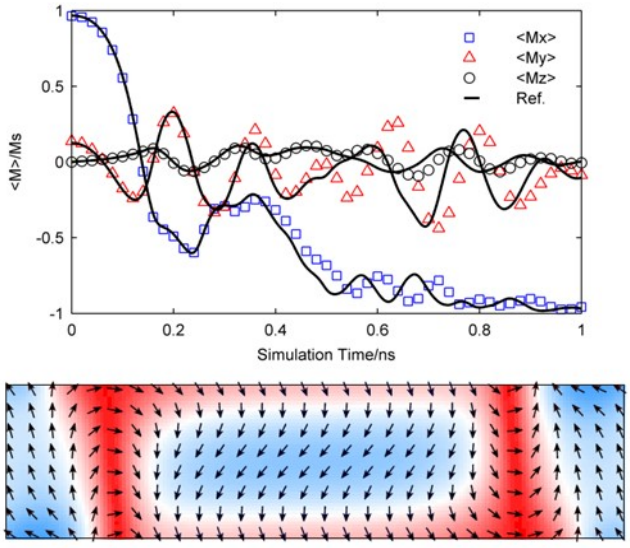

Fig. 3. The dynamics of averaged magnetization and a snapshot of the magnetization when averaged $\mathbf{M}_x$ first becomes zero, as specified by µMag standard problem 4b. In the top figure, results from our solver are marked by colored symbols, while the reference results are solid black lines.

2) Domain Wall Motion

Another test example is a nanowire with size 6.4 µm x 20 nm x 20 nm simulated with the tensor approach. A head-to-head domain wall is initialized at the center of the nanowire. An external field $\mu_0\mathbf{H}_x$ = -35.5 mT is applied to the nanowire so that the domain wall is expected to be driven in the –x direction according to the following equation [31]

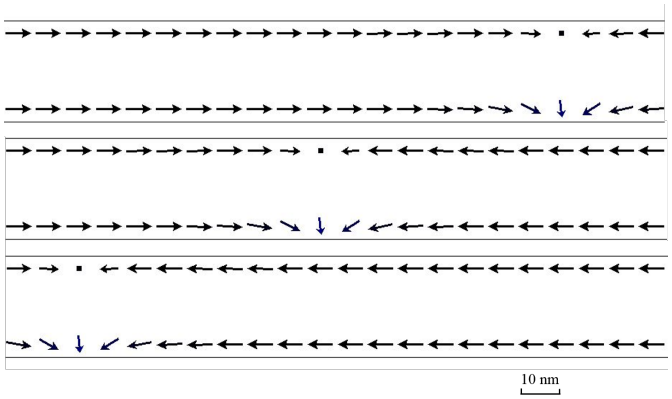

Fig. 4. Snapshots of domain wall in the nanowire at T = 3.40 ns, T = 4.08 ns and T = 4.76 ns, from top to down, respectively. The length-scale is also shown.

$$v = \frac{\gamma \cdot \Delta}{\alpha} H_{ext} \quad , \tag{12}$$

where $\Delta \approx A^{1/2}(K + 1/2\,\mu_0 M_s^2)^{-1/2}$ is the domain wall width and H_{ext} is the magnitude of external field. We used γ = 2.211e5 m/As, α = 0.5, A = 1.3e-11 J/m, M_s = 8e5 A/m and K = 1.0e5 J/m^3, which led to an estimated velocity of 63.56 nm/ns.

Fig. 4 shows snapshots of the moving domain wall. According to our test results using the tensor approach, the speed of the domain wall is 69.06 nm/ns. The simulated result is in close agreement to the predicted results.

B. *Finite Difference Solver on GPU*

1) *Accuracy Analysis*

Fig. 5 compares the accuracy of the magnetostatic field evaluation via the scalar potential and tensor approaches. In Fig. 5(a), which shows the accuracy of the magnetostatic field evaluation, we discretize a fixed-size cube (49 nm × 49 nm × 49 nm) into an increasing number of brick cells. The magnetization state $\mathbf{M} = z\,\hat{\mathbf{z}}$ was chosen so that both volume and surface charge densities exist ($\rho_M = -1\text{A/m}^2$, $\sigma_M = 49\text{A/m}$ when $z = 49$ nm, else $\sigma_M = 0$). For this case the magnetostatic field can be found analytically. The error was defined as

$$Error = \sqrt{\frac{\sum_i^N \left|\mathbf{H}_{\text{ms,i}}^{num} - \mathbf{H}_{\text{ms,i}}^{ana}\right|^2}{\sum_i^N \left|\mathbf{H}_{\text{ms,i}}^{ana}\right|^2}} \quad . \tag{13}$$

The "num" and "ana" in Eq. (13) are for "numerical method" and "analytical method" respectively.

Figure 5(b) shows the accuracy of the magnetization states obtained via the FDM solvers with the scalar potential and tensor approaches for the magnetostatic field. A fixed size cube (100 nm × 100 nm × 100nm) was discretized into an increasing number of brick cells. The magnetic parameters (γ = 2.211e5 m/As, α = 0.5, A = 4.91e-11 J/m, M_s = 7.96e5 A/m and K = 3.98e4 J/m3) were chosen to form a vortex magnetization state at equilibrium, which was sampled at uniformly distributed locations to evaluate the accuracy. The error was defined as in Eq. (13), with the magnetostatic field replaced with the magnetization and the analytical field replaced by the magnetization obtained as an "asymptotic" numerical solution for a very fine discretization.

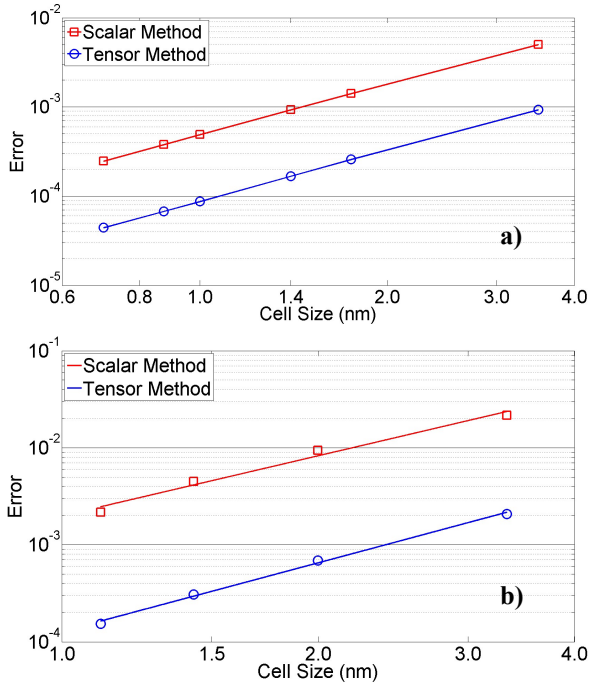

Fig. 5. The numerical error in the GPU implementation for a) magnetostatic field and b) Finite Difference solver with scalar potential and tensor approaches as a function of the discretized grid cell size. Both methods show quadratic convergence.

From Fig. 5 it is evident that both the scalar potential and tensor approaches have quadratic convergence. The tensor approach is more accurate by a factor because it avoids the approximation of charges at the boundaries and the numerical derivative operations in the superposition integrals. The reduced accuracy makes the tensor approach faster for the same error level. However, the overall micromagnetic solver error is determined not only by magnetostatics but also e.g. by exchange, which may have a higher error, especially for irregularly shaped structures. In such cases, the scalar potential approach is appealing because of its faster performance and smaller memory consumption, especially for large-scale problems. We further note that the magnetostatic and exchange fields can be calculated on different grids with different cell sizes. This approach allows reducing the error of the magnetostatic field calculation (by reducing the corresponding cell size) without reducing the time step related to the cell size used for the exchange field calculation [32].

2) *Speed Comparison*

The timing results of computing the magnetization time evaluation via the GPU and CPU implementations of OOMMF are shown in Table I. The computation of magnetostatic field, anisotropy field, exchange field and applied field are included. The CPU OOMMF code was compiled with all possible optimization flags to achieve the maximal performance for best comparison. (Note, however, that all OOMMF results are with double-precision floating

point operations, because OOMMF does not support single-precision operation. The GPU results in Table I are for single-precision operations. GPU results with double-precision operations are presented in Table II.) We considered cubic magnetic domains with different number of discretized cells as test cases. The number of cells in each dimension ranges from 16 to 128. One can see that the GPU OOMMF code has an increasing speed-up with respect to the total number of cells as compared to the CPU code. The increasing speed-up is because the GPU still has enough parallelization power for the problem scale we considered. The absolute performance is also high. In particular, for a problem with 2 million grid cells, compared to the time of 1322.9 ms by single-core CPU OOMMF, the GPU running time per single time step are 17.4 ms and 35.2 ms with the scalar and tensor approaches,

TABLE I
TIMING RESULTS OF FINITE DIFFERENCE SOLVER WITH FIXED TIME STEPPING

N	OOMMF		Finite Difference Solver			
	1 core (ms)	6 cores (ms)	Scalar (ms)	Speed -up	Tensor (ms)	Speed -up
4K	1.63	0.66	0.13	12.9	0.14	11.8
32K	14.11	5.12	0.34	41.4	0.60	23.7
256K	155.3	48.71	2.16	71.8	4.30	36.1
2M	1323	401.8	17.36	76.2	35.2	37.5

respectively. This corresponds to a 76.2x and 37.5x GPU-CPU speed-up. The CPU OOMMF running on 6 cores has the parallelization efficiency of 41%-55%. In particular, the for a problem with 2 million grid cells, the computational time of OOMMF on 6 cores is 401.82 ms.

Fig. 6 compares the speed of the GPU version of the solver implemented with scalar and tensor approaches. The running time per time step of both methods follows $O(N \log N)$ trend. We can also observe that the speed of the scalar approach is higher. At the points where cuFFT has highest efficiency on the GPU, e.g. 8^3, 16^3, 32^3, 64^3, 128^3, the scalar approach is about 2x faster. The speed-up is higher at other points because

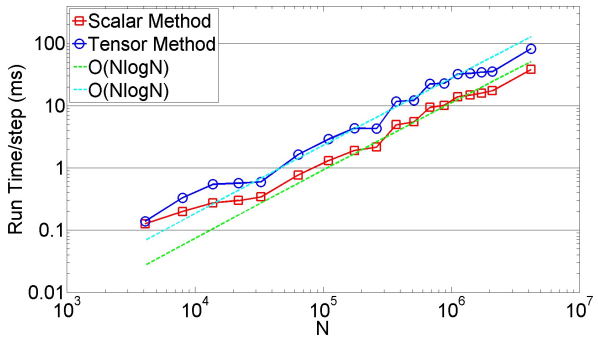

Fig. 6. Run time on the GPU per time step for the scalar potential and tensor methods as a function of the number of discretization cells N. The time scales as *O(NlogN)* for both approaches but the scalar potential approach is about two times faster.

cuFFT is working with a lower efficiency at these points, which leads to a higher weight of the FFT as compared to the other parts of the code. Since cuFFT performs better when the transformed array sizes are composed of small prime factors, such as 2, 3, 5 and 7, we zero-pad the array to these sizes. As a result, a smooth timing performance is achieved (Fig. 6). The presented implementation allows fitting up to 4M cells in 2GB GPU memory. Currently GPUs with 12 GB memory are available, which would allow problems up to 24M cells.

C. OOMMF on GPU

Next we present full simulation results of OOMMF implemented on GPU. The magnetostatic field is computed using the tensor approach for all cases. The CPU results are double-precision because OOMMF does not support single-precision operation. The GPU results are given for single- and double-precision.

1) Single Precision Performance

Fig. 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 also 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

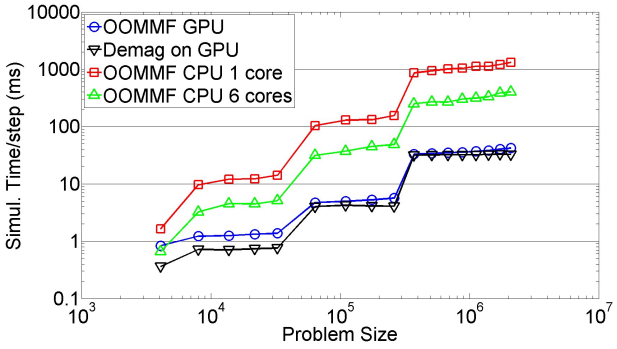

Fig. 7. Run time per time step for OOMMF on CPU and GPU as a function of the number of discretization cells N. The time for the magnetostatic field computation on the GPU is also included. Significant performance improvement is observed. The computation for the magnetostatic field takes most of the run time in the GPU implementation.

FFT array is padded to 128 although a size of only 2x33-1 = 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. 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.

Fig. 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.

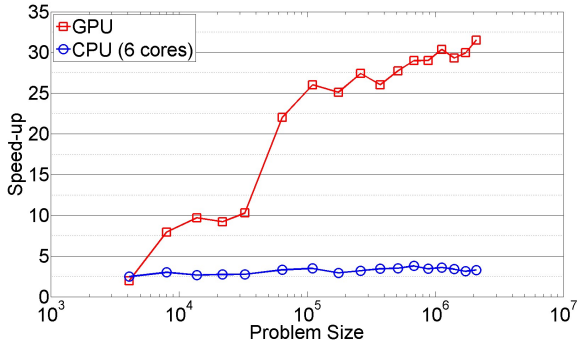

Fig. 8. GPU and multi-core CPU speed-up of OOMMF implementation as a function of the number of discretization cells N. An increase in the speed-up with N is observed.

2) *Double Precision Performance*

We also tested double-precision computations. The double-precision results are shown in Table II. 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 II
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

V. CONCLUSIONS

This paper presents a GPU-based FDM solver for the LLG equation in a micromagnetic context. Two approaches for computing the magnetostatic field were implemented. The first more conventional approach uses tensor products without evaluating differential operators. The second approach, presented for the first time, proceeds in three steps: the computation of volume and collocated surface charges, evaluating the magnetic scalar potential, and finding the field via the gradient of the scalar potential. Both approaches use FFTs for computing the resulting superposition summations, resulting in a computational cost of $O(N \log N)$. Both approaches for computing the magnetostatic field as well as the computation of all other LLG solver components were implemented on GPU. It is demonstrated that with proper implementation, the computational time for the exchange field can be substantially lower than that for the magnetostatic field, which is critical for the scalar potential approach and for future implicit time integration schemes.

The developed codes were integrated with the OOMMF simulator [5]. The integration is mostly seamless to the user in that the OOMMF interface remains the same and only the internal kernels were GPU-accelerated. The developed implementation shows an up to 32x GPU-CPU speed-up as compared to a fully optimized CPU version of OOMMF. The GPU version of the OOMMF code will be made available to the broad community of users at the OOMMF web site [20], providing opportunities to accelerate scientific research in magnetics. Future code updates will include options for different magnetostatic field approaches and new physics as well as further speed increases and increases to accessible problem sizes.

ACKNOWLEDGEMENT

Support from the National Institute of Standards and Technology (NIST) is gratefully acknowledged.

DISCLAIMER

The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology. All trademarks mentioned herein belong to their respective owners.

REFERENCES

[1] R. Chang, S. Li, M. Lubarda, B. Livshitz, and V. Lomakin, “FastMag: Fast micromagnetic simulator for complex magnetic structures,” *Journal of Applied Physics*, vol. 109, pp. 07D358, 2011.

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

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

[4] 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.

[5] 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..

[6] 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.

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

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

[9] 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.

[10] 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.
[11] M. Frigo and S.G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, Issue 2, pp. 216–231, 2005.
[12] Intel Corp. http://software.intel.com/en-us/intel-mkl/.
[13] CUDA CUFFT Library, Version 5.5. NVIDIA Corp., 2013.
[14] T.Sato, Y. Nakatani. "Fast micromagnetic simulation of vortex core motion by GPU." Journal of the Magnetics Society of Japan, vol. 35, Issue 3, pp. 163-170, 2011.
[15] A. Vansteenkiste, and B. Van de Wiele, "MuMax: a new high-performance micromagnetic simulation tool." *Jounal of Magnetism and Magnetic Materials*, Vol. 323, Issue 21, pp. 2585-2591, 2011.
[16] S. Fu, R. Chang, S. Couture, M. Menarini, M.A. Escobar, M. Kuteifan, M. Lubarda, D. Gabay and V. Lomakin, "Micromagnetics on high-performance workstation and mobile computational platforms", *Journal of Applied Physics,* vol. 117, no. 17, p. 17E517, 2015.
[17] 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, issue 4, pp. 469-476, 2010.
[18] 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, Issue.8, pp. 5816-5818, 1999.
[19] 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.
[20] "The Object Oriented MicroMagnetic Framework (OOMMF) project at ITL/NIST", http://math.nist.gov/oommf/.
[21] Z. Li, S. Zhang. "Magnetization dynamics with a spin-transfer torque." Physical Review B, vol. 68, issue. 2, pp. 024404, 2003.
[22] 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.
[23] C. Abert, E. Lukas, G. Selke, A. Drews, T. Schrefl. "Numerical methods for the stray-field calculation: A comparison of recently developed algorithms." Journal of Magnetism and Magnetic Materials, vol. 326, pp 176-185, 2013.
[24] J.E. Miltat, M.J. Donahue, "Numerical micromagnetics: Finite difference methods." Handbook of magnetism and advanced magnetic materials, 2007.
[25] K. Ramstöck, T. Leibl, and A. Hubert. "Optimizing stray field computations in finite-element micromagnetics." *Journal of magnetism and magnetic materials*, vol. 135, issue 1, pp. 97-110, 1994.
[26] Nvidia, *CUDA C Programming Guide*, version 5.5, 2013.
[27] 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.
[28] M. Harris, "Optimizing parallel reduction in CUDA", Nvidia Developer Technology 2, Nvidia Corp., 2007.
[29] muMAG Micromagnetic Modeling Activity Group, http://www.ctcms.nist.gov/~rdm/mumag.org.html.
[30] L. D. Buda, I. L. Prejbeanu, M. Demand, U. Ebels and K. Ounadjela, "Vortex states stability in circular Co(0001) dots", The 8th Joint Magnetism and Magnetic Materials - Intermag Conference Proceedings, 2001.
[31] G.S.D. Beach, C. Nistor, C. Knutson, M. Tsoi, and J.L. Erskine, "Dynamics of field-driven domain-wall propagation in ferromagnetic nanowires", *Nature Materials,* vol. 4, pp. 741-744, 2005.
[32] S. Fu, L. Xu, V. Lomakin, A. Torabi, B. Lengsfield, "Modeling Perpendicular Magnetic Multilayered Oxide Media with Discretized Magnetic Layers,", Magnetics, IEEE Transactions on, vol. 51, no.11, pp.1-4, 2015.