Development of a Highly Parallelized Micromagnetic Simulator on Graphics Processors

Aamir Ahmed Khan*, Paolo Lugli
Institute for Nanoelectronics,
Technische Universität München, Germany
Email: *aamir@mytum.de

Wolfgang Porod, György Csaba
Center for Nano Science and Technology,
University of Notre Dame, Notre Dame, IN, USA

***Abstract*—This work introduces the development of a micromagnetic simulator running on highly parallel Graphics Processing Units (GPU's). We show how main components of the calculation (ODE solver and magnetic potential calculation) can be implemented to provide optimal speedup for various problem sizes. For smaller problems, direct potential calculation is the most efficient method, while large problems are best handled by a fast multipole algorithm.**

***Index Terms*—Micromagnetic simulations, graphics processors, parallel algorithms, fast multipole method, nanomagnet logic.**

I. Introduction

Engineered magnetic nanostructures are emerging as new, promising ways to realize nanoscale electronic devices, such as Nanomagnet Logic [1]. Research is flourishing on experimental and computational micromagnetics [2].

Micromagnetics is a well-established theory and gives excellent agreement with experimental observations. However, micromagnetic simulations are computationally intensive and offer very few computational shortcuts that could make the simulations more efficient. This is due to the high nonlinearity of micromagnetic equations and the wide range of length and time scales involved. In order to precisely represent the geometry of nano-patterned structures and the fine structure of domain walls the spatial discretization of the magnetic material should go all the way down to a few nanometers. The entire size of a nanomagnetic devices could be several micrometers. Magnetization dynamics span a wide time scale from 10^{-14} seconds to microseconds. This motivates the construction of a high-performance micromagnetic solver.

Several state of the art micromagnetic solvers offer some level of parallelization and there are micromagnetic codes running on parallel supercomputers. Performance scaling is far from optimal for most micromagnetic problems, mainly due to the non-local nature of magnetic field calculations, which makes domain decomposition difficult.

Graphics Processing Units (GPU's) offer the best price/performance ratio for solving numerical problems [3]. In GPU's, a large number (several hundreds) of cores concurrently compute a mathematical function over a large grid of simulation cells. The NVIDIA Tesla C2050 card we used contains 448 processing units a 3GB internal memory and sells below USD 2000. This motivates the development of a GPU-based micromagnetic simulator. We are aware of similar efforts from other groups as well [4].

The next section of our paper will outline the flow of a generic micromagnetic simulation. By far the most time consuming part of a micromagnetic simulation is the calculation of long-range magnetic fields (magnetic scalar potential). A direct method for calculating magnetic fields scales very badly with problem size ($O(N^2)$), where N is the number of mesh elements), but due to the sheer number of available GPU cores it is an optimal solution for small problems. We introduce a GPU-accelerated simulator based on this method in Section III. A fast multipole method (FMM, Sec. IV) is our choice for large-sized problems. The final section (Sec. V) benchmarks the first version of our simulator and discusses future work for optimization of algorithms to maximize the speedup.

II. Components of a Micromagnetic Simulator

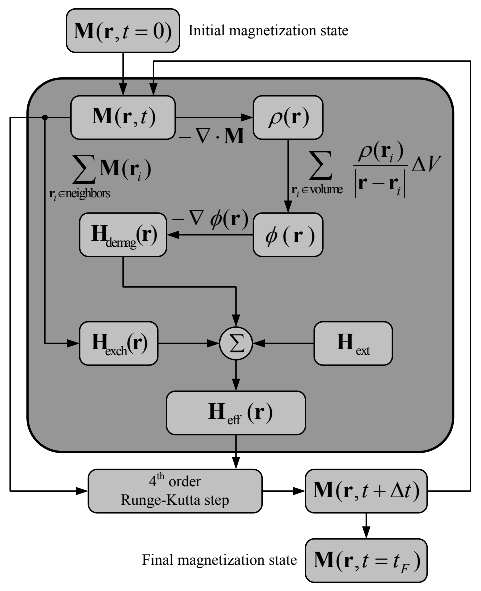
The goal of a micromagnetic simulation is to compute the time-evolution of an $\mathbf{M}(\mathbf{r}, t)$ magnetization distribution by solving Landau-Lifshitz equation. The change of the $\mathbf{M}(\mathbf{r}, t)$ magnetization is due to the effective magnetic field:

$$\mathbf{H}_{\text{eff}} = \mathbf{H}_{\text{exch}} + \mathbf{H}_{\text{demag}} + \mathbf{H}_{\text{ext}}, \tag{1}$$

which is a superposition of 'real' magnetic fields (such as the $\mathbf{H}_{\text{demag}}$ demagnetization field and the $\mathbf{H}_{\text{ext}}$ external field, coming from the solution of Maxwell's equations) and fields that represent quantum-mechanical forces in a quasi-classical way (such as the $\mathbf{H}_{\text{exch}}$ exchange field). The most essential component of the micromagnetic simulator is the calculation of this effective field.

There are a number of publications describing the construction of a micromagnetic simulator [2]. Figure 1 summarizes the main steps of the simulation flow.

The magnetization distribution is represented on a three-dimensional mesh, with a typical mesh size ranging from 2 to 10 nm. Depending on the simulated volume, the number of mesh elements can be anywhere from $N = 10^2$ to $N = 10^8$. In each iteration step, the effective magnetic field is evaluated on all lattice points and the change of $\mathbf{M}$ is calculated by an explicit time-integration method. The iteration is continued until a stationary $\mathbf{M}(\mathbf{r}, t)$ is reached, which is typically the desired final result of the simulation.



$\mathbf{M}(\mathbf{r}, t=0)$
Initial magnetization state
$\mathbf{M}(\mathbf{r}, t)$
$-\nabla\cdot\mathbf{M}$
$\rho(\mathbf{r})$
$\sum_{\mathbf{r}_i\in\text{neighbors}}\mathbf{M}(\mathbf{r}_i)$
$\sum_{\mathbf{r}_i\in\text{volume}}\frac{\rho(\mathbf{r}_i)}{|\mathbf{r}-\mathbf{r}_i|}\Delta V$
$-\nabla\phi(\mathbf{r})$
$\mathbf{H}_{\text{demag}}(\mathbf{r})$
$\phi(\mathbf{r})$
$\mathbf{H}_{\text{exch}}(\mathbf{r})$
Σ
$\mathbf{H}_{\text{ext}}$
$\mathbf{H}_{\text{eff}}(\mathbf{r})$
4th order Runge-Kutta step
$\mathbf{M}(\mathbf{r}, t+\Delta t)$
Final magnetization state
$\mathbf{M}(\mathbf{r}, t=t_F)$

Fig. 1. Flow chart of a micromagnetic simulation.

III. A GPU-based Solver with Direct Potential Calculation

A. *Components of the effective field*

The exchange interaction is local – its magnitude at given lattice point depends only on the magnetization in the immediate neighborhood of this point. It can be approximated by the six-neighbor sum:

$$\mathbf{H}_{\text{exch}} = \frac{2A_{\text{exch}}}{M_s^2\mu_0\Delta x^2}\sum_{i\in\text{neighbors}}\mathbf{M}(\mathbf{r_i}), \qquad (2)$$

where Δx is the cell sizes of the cubic numerical mesh [2] and A_{exch} is a physical constant characterizing the strength of the exchange interaction.

The demagnetizing field describes the long-range magnetostatic field created by the $\mathbf{M}(\mathbf{r}, t)$ distribution. The calculation of demagnetization field starts by calculating the magnetic volume charges over the entire mesh using the divergence operation:

$$\rho(\mathbf{r}) = -\nabla\cdot\mathbf{M}(\mathbf{r}), \qquad (3)$$

then evaluating the magnetic scalar potential at each lattice point:

$$\phi(\mathbf{r}) = \frac{\mathbf{1}}{\mathbf{4}\pi}\sum_{\mathbf{r_i}\in\text{volume}}\frac{\rho(\mathbf{r_i})}{|\mathbf{r}-\mathbf{r_i}|}\mathbf{\Delta V} \qquad (4)$$

and finally calculating the magnetic field as the gradient of the scalar potential:

$$\mathbf{H}_{\text{demag}} = -\nabla\phi(\mathbf{r}) \qquad (5)$$

Clearly, the evaluation of eq. 4 is most time consuming part of the calculation. It represents all the pairwise interactions between cells and requires $O(N^2)$ steps. All other steps are local and scale with $O(N)$.

For magnetic field calculations, one generally has to take into account not only volume charges but also surface charges, which appear at the boundaries of magnetic bodies and where eq. 3 cannot be interpreted, due to the discontinuity of the magnetization. To keep the algorithm simple, we did not explicitly consider surface charge contributions, but instead we interpreted the discretized form of eq. 3 over the entire volume. This 'smears' an equivalent volume charge on the cells adjacent to the surface. There is some error in the magnetic field at the boundaries of magnetized regions, but this error becomes negligible on the neighboring lattice sites.

The calculation of the magnetic charges and field gradients requires that there is at least one padding layer around the magnetization distribution (i.e. cells where $M = 0$).

B. *Time stepping algorithm*

We solved the Landau-Lifshitz equations by a fourth-order Runge Kutta method with adaptive time stepping. The criteria for finding an acceptable time step was to keep the change on the magnetization ($\Delta t\cdot\max(\frac{\mathbf{M}\times\mathbf{H}}{M_s^2})$) smaller than a pre-defined limit 10^{-13}. We found that this time stepping method often introduces spurious oscillations. To prevent these artifacts we reject any time step that would increase the sum of demagnetization and exchange energies (integrated for the entire volume). This is similar to the criteria used in well established OOMMF code [5]. We found that the performance of the Runge-Kutta method (which requires four full potential calculations per time step) is still superior compared to simple Euler scheme (that requires only a single field update).

C. *Parallelization of the field solver*

Both the ODE solver and the exchange field calculation can be parallelized by domain decomposition. The ODE solver alone can be sped up by a factor of 100 on the used GPU. However, for any non-trivial sized problem ($N > 100$) the potential calculation eq. 4 is by far the most time consuming part ($> 90\%$) and so we focused on the GPU implementation of this part. All the local operations including the ODE solver are parallelized only on the CPU using OpenMP.

D. *Parallelization of direct potential calculation*

For pairwise interactions, there are two nested loops in the direct method – sources and observers, and their order is interchangeable. The NVIDIA GPU's also offer two levels of parallelism – multiprocessor (block) level and thread level, so this problem maps straightforwardly to the GPU. At the observer level, we have to sum potential contributions from N sources; and to execute for each source, a *read-modify-write* instruction of the form,

$$\phi[\text{observer}_i] = \phi[\text{observer}_i] + \frac{q[\text{source}_j]}{r_{ij}} \qquad (6)$$

This read-modify-write instruction requires synchronization among parallel workers. We mapped sources to threads and observers to blocks because there is no efficient way to share data among the blocks, while doing so among the threads is highly efficient. For summing up potential from N threads, parallel reduction algorithm [6] is utilized, which exploits architectural features of the GPU efficiently.

IV. Parallel Potential Calculation Based on the Fast Multipole Method

A. *Implemention of the fast multipole method*

Fast multipole method (FMM) [7] enables to efficiently perform the summation of eq. 4. This was already exploited for micromagnetic simulations [8]. The underlying idea is that the magnetic charges can be grouped together and the far field of this cluster at any far-lying point in space can be approximated with only a few terms of the multipole expansion, regardless of the distribution of charges in the cluster. The computational domain is hierarchically subdivided into four equal parts starting from whole domain (level 0) and going down to finer levels. At refinement level L, there are 4^L cells each of size $\frac{N}{4^L}$, where N is the number of mesh elements in the domain. Multipole coefficients for a given cell are calculated from the charge distribution inside the cell:

$$M_l^m = \sum_{i=1}^{K} q_i r_i^l Y_l^{m*}(\theta_i, \phi_i), \tag{7}$$

and then used to evaluate potential at far-lying cells by the multipole expansion [7] [8]:

$$\phi(\mathbf{r}) \approx \sum_{l=0}^{P} \sum_{m=-l}^{l} \frac{M_l^m}{r^{l+1}} Y_l^m(\theta, \phi), \tag{8}$$

where $Y_l^m(\theta, \phi)$ are spherical harmonics as defined by [7], and can be derived from the corresponding associated Legendre polynomials. P is the truncation of the multipole series (typically $P = 3$ gives sufficient accuracy).

Figure 2 illustrates that at a given hierarchy level L in a 2D computational domain, which potential contributions are evaluated using eq. 8. They are to the not-immediate neighbors that have not yet been considered in the previous coarser hierarchy levels. At a given level of hierarchy there are at most 27 such cells around each subdomain source.

The origin of the speedup (as compared to direct summation) is that the interaction between far-lying mesh points are evaluated using multipole expansions on large subdomains. Interactions between closer-lying regions are accounted for with multipole expansions at finer levels of hierarchy and using smaller subdomains for the multipole expansion. Only potential contributions from immediate neighbors at the finest level are calculated with pairwise summation.

As nanomagnet logic devices are planar structures, we implemented the FMM algorithm in 2D. The hierarchical subdivisions are applied separately to each planar layer and the potential contributions from each layer are summed up to the 3D volume. The method scales with $O(N_{\text{layer}}^2 N_{\text{plane}} \log N_{\text{plane}})$, where N_{layer} is the number of layers and N_{plane} is the number

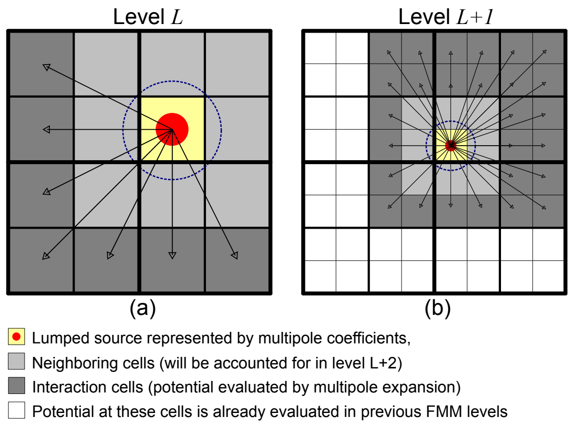

Level L
Level $L+1$
(a)
(b)
Lumped source represented by multipole coefficients,
Neighboring cells (will be accounted for in level L+2)
Interaction cells (potential evaluated by multipole expansion)
Potential at these cells is already evaluated in previous FMM levels

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

of mesh elements per plane. For typical structures $N_{\text{layer}} \approx 10$, $N_{\text{layer}} \approx 10^6$), this seems to be the optimal solution.

The FMM method carries a significant overhead. It is time-consuming to calculate the $(P+1)^2$ multipole coefficients for each domain. It requires careful indexing and record-keeping to consider all interactions exactly once and exactly at the appropriate level of hierarchy. Thus FMM is justified only for relatively large problems.

B. *Paralellization of the fast multipole method*

Straightforward complexity analysis shows that running time of potential evaluation through multipole expansion (using eq. 8) is at least $27N_{\text{layers}}$ times greater than that of coefficient calculation. So as a first step we paralellized only this part of the FMM algorithm. For work sharing scheme, it is natural to distribute work over 27 observer cells to GPU blocks, since they do not have to share data among each other. Threads within a GPU block are then mapped to individual mesh points inside the corresponding observer cell for the evaluation of multipole expansion to contribute to their potential value.

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

V. Results, Benchmarking and Conclusions

A. *A simulation example*

The multi-domain to single-domain transition is one of the most characteristic phenomena of nanomagnetic behavior. For small magnets, exchange fields drive the magnet into a homogeneously magnetized (single-domain) state. For larger magnets, the stronger demagnetization field promotes the formation of a vortex state. The transition between the two states takes place at a well defined magnet size. This can be used to test our micromagnetic simulator and make comparison with established codes.

We investigated a 5 nm thick, square-shaped permalloy nanomagnet. The energy difference between a relaxed vortex and a relaxed single domain state ($\Delta E = E_{1\text{domain}} - E_{\text{vortex}}$) is shown in fig. 3, as a function of the magnet size. The phase transition occurs at 180 nm, in good agreement with results given by well-established codes, such as OOMMF [5].

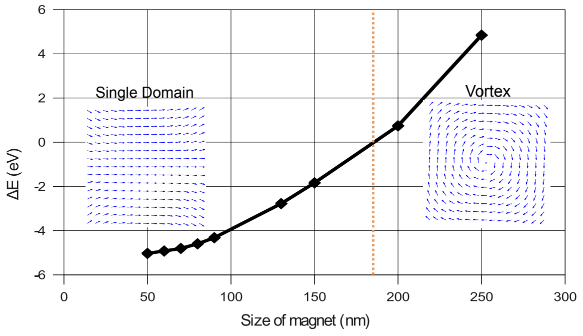


Fig. 3. Energy difference between a relaxed single-domain and vortex state of a square-shaped permalloy nanomagnet.

We also verified that the parallel (GPU) and the CPU-based codes are giving the same results within floating-point accuracy.

B. Benchmarking the parallelized code

The direct potential calculation is straightforward to paraellize as we discussed in Section III. Speedup factors (as compared to the single-threaded CPU version) are shown in fig. 4. As other components of the micromagnetic solver are running on the CPU, the GPU solver has to be launched in each iteration step of the time loop. Speedups up to 30× are realized when, for large problems, the GPU launch time becomes negligible compared to the computation time.

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

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

The parallel FMM shows its strength for large-sized problems ($N > 2 \cdot 10^5$), as demonstrated in fig. 5, where it steeply outperforms the direct calculation. For small problems, however, the well-paralellized direct calculation does better.

To summarize, we were introducing our ongoing work on a high-performance GPU-accelerated micromagnetic simulator. The simulator is tested on a number of micromagnetic problems and shows significant speedups. Work is in progress to implement the FMM-based micromagnetic solver entirely on the GPU and eliminate the memory bottleneck of the parallel FMM implementation.

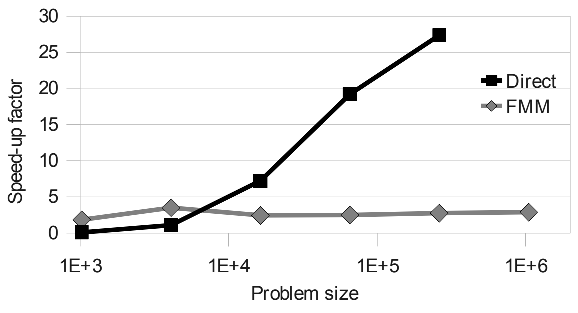


Fig. 4. Comparisons of the parallel (GPU) and single-threaded versions of the potential calculations. Direct method has comparative high speed-up because of straightforward parallelization as opposed to FMM case.

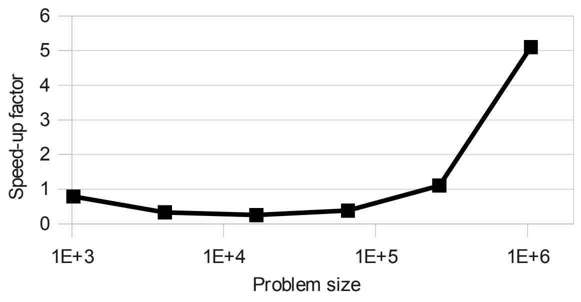


Fig. 5. Speedup of the GPU based FMM method vs. the GPU based direct method.

Acknowledgments

We acknowledge financial support of Deutsche Forschungsgemeinschaft (DFG) in the framework of the project 'Field-coupled Circuits in Magnetic Multilayers' and of the Office of Naval Research (ONR).

References

[1] A. Imre, G. Csaba, L. Ji, A. Orlov, G. H. Bernstein, W. Porod, Majority Logic Gate for Magnetic Quantum-Dot Cellular Automata, Science 13 Vol. 311. no. 5758, pp. 205 – 208 (2006)
[2] J. E. Miltat, M. J. Donahue, Numerical Micromagnetics: Finite Difference Methods Handbook of Magnetism and Advanced Magnetic Materials. Edited by Helmut Kronmüller and Stuart Parkin. Volume 2: Micromagnetism. 2007 John Wiley & Sons
[3] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, GPU Computing, Proceedings of the IEEE, 96, no.5, pp.879-899, May 2008.
[4] S. Li, B. Livshitz, V. Lomakin, Graphics Processing Unit Accelerated $O(N)$ Micromagnetic Solver IEEE Transactions On Magnetics, 46, 6, June 2010
[5] The Object Oriented MicroMagnetic Framework (OOMMF) is a widely used, public-domain micromagnetic software micromagnetic program. http://math.nist.gov/oommf/
[6] M. Harris, Optimizing parallel reduction in CUDA, NVIDIA CUDA SDK – Data-Parallel Algorithms, 2007
[7] L. Greengard, The Rapid Evaluation of Potential Fields in Particle Systems, MIT Press, Cambridge (1988).
[8] G. Brown, M. A. Novotny, and P. A. Rikvold, Langevin simulation of thermally activated magnetization reversal in nanoscale pillars Phys. Rev. B 64, 134422 (2001).