Fast Boundary Methods for Magnetostatic Interactions in Micromagnetics

Hermann Forster, Thomas Schrefl, Rok Dittrich, Werner Scholz, and Josef Fidler

***Abstract*—Fast boundary integration methods are implemented to increase the efficiency of the calculation of the magnetostatic interaction of magnetic particles. We test the treecode algorithm and a hierarchical matrices method to improve the original used matrix-vector multiplication with a fully populated matrix. We compare the CPU-time used for the setup phase and for the matrix-vector multiplication, as well as the required storage.**

***Index Terms*—Fast boundary methods, finite elements, hierarchic matrices, micromagnetics, treecode.**

I. Introduction

MICROMAGNETIC simulations of realistic magnetic devices require the calculation of the magnetostatic interactions between distinct magnetic parts. Hybrid finite element/boundary element (FE/BE) algorithms as originally proposed by Fredkin and Koehler are very efficient, since they require no mesh between the particles [1]. This method is described in Section II. In addition, the FE/BE discretization allows arbitrarily shaped structures. However, the BE part of this algorithm leads to a fully populated matrix of size $\mathcal{O}(N^2)$, where N is the number of boundary nodes. As a consequence, storage and CPU-time scale with N^2, which causes performance problems in structures with high aspect ratio where most nodes are on the boundary. To overcome this problem, various techniques to accelerate the BE method have been proposed. In this paper, we compare three methods:

1) fully populated boundary matrix (Section II);
2) treecode method as used for particle simulations (Section III-A);
3) hierarchical matrices built by using the ACA-algorithm (Section III-B).

II. Hybrid FE/BE Method

Within the framework of the FE/BE method, the magnetic scalar potential is calculated as sum $U = U_1 + U_2$ with

$$\Delta U_1 = \nabla \cdot \mathbf{J}(\mathbf{r}) \quad \text{for } \mathbf{r} \in \omega_{\mathrm{m}} \tag{1}$$

$$U_2(\mathbf{r}) = \frac{1}{4\pi} \int \frac{U_1(\mathbf{r}')(\mathbf{r}-\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|^3} \mathrm{d}\mathbf{n} + \left(\frac{\Omega(\mathbf{r})}{4\pi} - 1\right) U_1(\mathbf{r}') \quad \text{for } \mathbf{r} \in \Gamma \tag{2}$$

$$\Delta U_2 = 0 \quad \text{for } \mathbf{r} \in \omega_{\mathrm{m}}. \tag{3}$$

Manuscript received January 10, 2003. This work was supported by the Austrian Science Fund under Y132-PHY.

The authors are with the Institute of Solid State Physics, Vienna University of Technology, A-1040 Vienna, Austria (e-mail: thomas.schrefl@tuwien.ac.at).

Digital Object Identifier 10.1109/TMAG.2003.816458

The integral in (2) is over all the surfaces Γ of magnetic bodies ω_{m} and $\Omega(\mathbf{r})$ is the solid angle subtended by Γ at $\mathbf{r}$. Equation (1) can be solved by a standard FE method. The discretization of (2) leads to a matrix vector multicplication

$$U_2^j = B_{ij} U_1^i. \tag{4}$$

The matrix B depends only on the geometry and the FE mesh and thus has to be computed only once for a given FE mesh in the so-called setup-phase. However, B may change with time for problems including moving parts.

III. Fast Boundary Integration Methods

The application of the FE/BE method as explained in Section II has many advantages. Only the magnetic bodies of the considered domains need to be discretized, open boundary problems pose no additional difficulties, and problems including motion can be treated elegantly [2]. However, application of the FE/BE method leads to dense matrices. The storage requirements and computational costs are of $\mathcal{O}(N^2)$, where N is the number of unknowns, respectively, the number of boundary nodes, although the values of most of the elements in the boundary matrix B are small. Clearly, we see that we run into problems if we increase the size of the used model due to the limited computational power. Obviously, when dealing with linear systems of N equations, one has optimal efficiency if the computational amount of work is $\mathcal{O}(N)$. For many situations characterized by a sparse system matrix, optimal solution algorithms are known.

In this paper, two different fast boundary integration methods are used to get this advantage. This speeds up the matrix-vector multiplication $U_2 = BU_1$ of (4), with the boundary matrix B, which is originally dense.

A. Treecode

In tree algorithms, particles are arranged in a hierarchy of clusters. When the force on a particular particle is computed, the interaction exerted by distant groups is approximated by their lowest multipole moments. In this way, the computational cost for a complete force evaluation can be reduced to order $\mathcal{O}(N \log N)$ [3]. The forces become more accurate if the multipole expansion is carried out to higher order, but the increasing cost of evaluating higher orders might make it more efficient to terminate the multipole expansion and instead use a larger number of smaller tree nodes to achieve a desired force accuracy [4]. We will just use dipole approximations. This means we interpret each boundary element as a dipole having a dipole moment. So the particles in the magnetostatic problem are dipoles

sitting on the boundary elements. The arising interaction of the dipoles on a specific boundary node has to be calculated.

We use the Barnes and Hut [5] tree construction. In this scheme, the domain, which surrounds all dipoles, is hierarchically partitioned into a sequence of cubes, where each cube contains eight siblings, each with half the side length of the parent cube. These cubes form the nodes of an oct-tree structure. The tree is constructed such that each cube contains either exactly one dipole, or is parent to further cubes, in which case the parent cube carries the dipole moments of all the dipoles that lie inside this cube.

The computation of the interaction between the dipoles proceeds by walking through the tree and summing up appropriate contributions from the tree elements. In the tree walk, the dipole moment of a cell of size l is used only if

$$r > l/\theta \tag{5}$$

where r is the distance of the particular point to the center of mass of the cell and θ is an accuracy parameter. If a cell fulfills this criterion, the tree walk along this branch can be terminated, otherwise it is "opened," and the walk is continued with all its siblings. For smaller values of the parameter θ, the forces will in general become more accurate, but also more CPU-time consuming. $\theta = 0.2$ is a good choice for the tradeoff between the advantage in CPU-time due to the clustering and the loss in accuracy.

The tree construction can be made by inserting the dipoles one after the other in the tree. Once the grouping is completed, the multipole moments of each cube (if they are parent cells) can be recursively computed from the moments of its daughter cubes.

B. $\mathcal{H}$-Matrices

Using hierarchic matrices, the fullmatrix B is approximated by a class of matrices (called $\mathcal{H}$-matrices [6], where $\mathcal{H}$ abbreviates "hierarchical"). These matrices are not sparse in the sense that there are only few nonzero entries, but they are data sparse in the sense that these matrices are described by only few data. The class of $\mathcal{H}$-matrices contains certain sparse matrices and approximates very well full matrices as they arise from integral operators. It is proposed that the amount of work is almost linear in N [7].

The basic building blocks for $\mathcal{H}$-matrices are $\mathrm{R}k$-matrices which are low-rank matrices. These matrices form subblocks of the $\mathcal{H}$-matrix. For the determination of the $\mathrm{R}k$-matrices, we use the *adaptive cross approximation*, which approximates the matrix using an iterative scheme described in [8]. Again, there is an accuracy parameter ϵ. We suggest the use of $\epsilon = 0.001$.

Table I shows a comparison between the different boundary integration methods with varying accuracy parameters.

IV. Geometry

A model for the test of different fast boundary integration methods should involve two features:

1) interacting particles, since the FE/BE method does not need a mesh outside of the particles;
2) high aspect ratio, because the FE/BE method scales with N^2.

TABLE I
COMPARISON OF THE STRAY FIELD ENERGY E_{stray} OF A CUBE FOR DIFFERENT BOUNDARY INTEGRATION METHODS. THE ANALYTICAL VALUE IS $E_{\mathrm{stray}} = 0.166\,67$

fullmatrix	treecode		hierarchic mat.	
	θ	E_{stray}	ϵ	E_{stray}
0.16482	0.0	0.16630	0.00001	0.1645613
	0.2	0.16482	0.0001	0.1645613
	0.4	0.16802	0.001	0.1645624
	0.6	0.16918	0.01	0.1645580
	0.8	0.17091	0.1	0.1645029
	1.0	0.17286	1	0.1643937

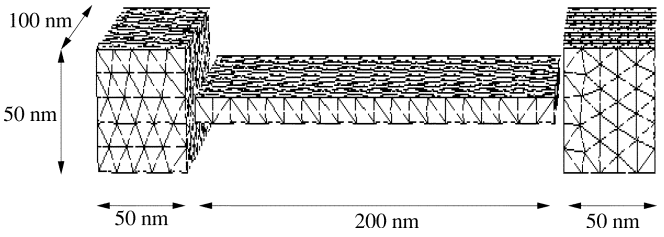

Fig. 1. Model system. Surface grid with 966 boundary nodes.

Therefore, a sensor element has been chosen as our test system. Fig. 1 shows the model. A soft magnetic film (thickness 10 nm) is stabilized by two permanent magnets. The air gap in between has a size of 5 nm. The permanent magnets consist of CoPt with $J_{\mathrm{s}} = 1.25$ T, an exchange constant of $A = 1.6 \times 10^{-11}$ J/m, and a magnetocrystalline anisotropy constant of $K_{\mathrm{u}} = 4 \times 10^5$ J/m. The soft magnetic sensor element in the middle is NiFe, where $J_{\mathrm{s}} = 1$ T, the exchange $A = 1.3 \times 10^{-11}$ J/m, and no anisotropy $K_{\mathrm{u}} = 0$.

To vary the number of boundary nodes, the model is remeshed with different mesh sizes between 50 and 1.75 nm. This results in 54 to 31 642 boundary nodes and 96 to 63 272 boundary elements, respectively. Fig. 1 shows the boundary mesh with 996 boundary nodes and 1920 boundary elements.

V. Results

The required storage and the CPU-time for the matrix-vector multiplication of (4) are compared. Since the boundary matrix B may change with time for problems including moving parts or changing meshes, we also consider the CPU-time required for the setup phase.

Fig. 2 shows the storage required during the simulation for different numbers of boundary nodes. The fullmatrix method shows the N^2 behavior. Obviously, this method is limited by memory and the computational power. The use of hierarchic matrices reduces the needed storage by 93%. The treecode method saves additional storage and reaches savings of 98% as compared with the fullmatrix method. The use of these alternative methods allows the simulation of larger systems.

CPU-time tests with the fullmatrix method are, therefore, only possible for $N < 10\,000$, because of the N^2 dependence and the limited memory in our workstation. Fig. 3 shows that both the treecode and the hierarchic matrices approach reduce the CPU-time of the setup phase as compared with the N^2

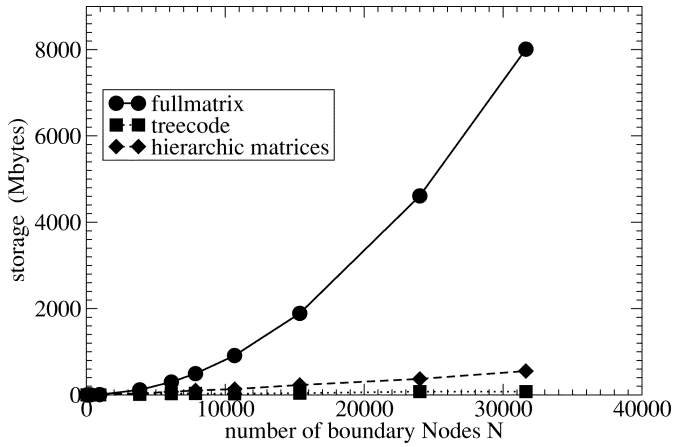

Fig. 2. Storage as a function of the number of boundary nodes.

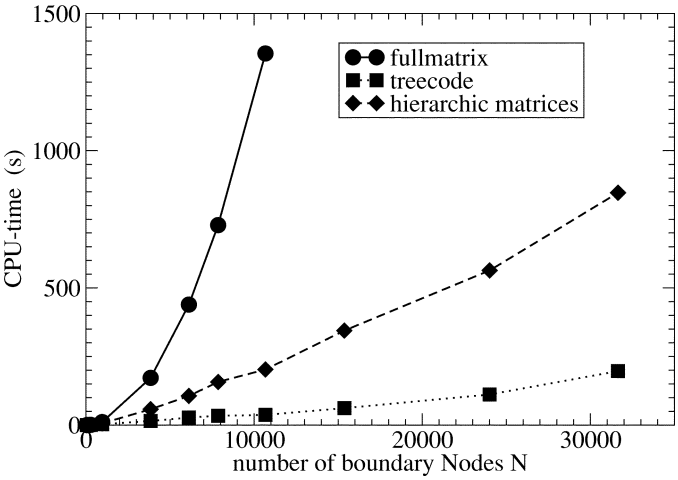

Fig. 3. CPU-time required for the setup phase.

dependence of the fullmatrix method. The setup phase involves all operations which are performed only once for a given mesh and are made before the simulation is started, e.g., the building of the fullmatrix, the tree, or the building of the hierarchic matrices. We see that the greatest acceleration can be reached with the treecode method. In our implementation, the hierarchic matrices are somewhere in the middle between the fullmatrix method and the treecode. This may be explained due to the use of the ACA algorithm for the building of the hierarchic matrices. The performance of this algorithm is not optimal. The implementation of the $\mathcal{H}^2$-matrices algorithm might reduce the required CPU-time for the building of the hierarchic matrices [9].

Finally, we evaluated the required CPU-time for 1000 matrix-vector multiplications according to (4) (Fig. 4). Even for a small problem size, $N = 10\,000$, the hierarchic matrices method reduces the CPU-time by a factor of 10 as compared with the multiplication of the fullmatrix. With increasing problem size, the speedup becomes more significant. The treecode scales

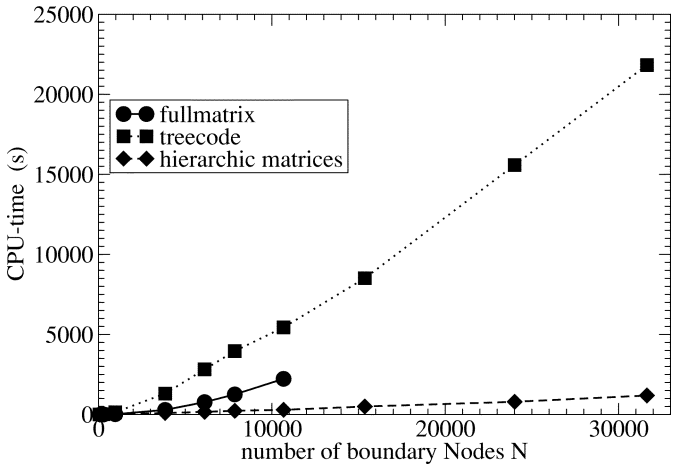

Fig. 4. CPU-time required for 1000 matrix vector multiplications.

with $N \cdot \log(N)$, but the very CPU-time expensive tree search increases the CPU-time for matrix-vector multiplications for small problems.

VI. Conclusion

Because of the N^2-dependence, the fullmatrix method is not suitable for large systems. Due to the very short setup-phase, the treecode method is recommended for moving parts, for example, recording simulations using a fully discretized head. Also, short setup-phases are required for changing meshes as resulting from a mesh refinement algorithm. Perhaps the use of the $\mathcal{H}^2$ matrices algorithm for building the hierarchical matrices reduces the CPU-time of the setup phase, too. Due to the time-consuming tree search, the hierarchic matrices method performs the matrix-vector multiplication in the fastest way, so, this is the method of choice for the solution of the Landau–Lifshitz equation on a fixed grid.

References

[1] D. R. Fredkin and T. R. Koehler, "Hybrid method for computing demagnetizing fields," *IEEE Trans. Magn.*, vol. 26, pp. 415–417, Mar. 1990.

[2] T. Schrefl, H. Forster, M. Schabes, and B. Lengsfield, "Finite element simulation of head/media interactions in perpendicular recording," in *Proc. Compumag*, J. Webb and D. Giannacopoulos, Eds., Saratoga Springs, FL, 2003.

[3] A. W. Appel, "An efficient program for many-body simulations," *SIAM J. Sci. Stat. Comp.*, vol. 6, pp. 85–103, 1985.

[4] S. L. W. McMillan and S. J. Aarseth, "An $\mathcal{O}(n \log n)$ integration scheme for collisional stellar systems," *Astrophys. J.*, vol. 414, p. 200, 1993.

[5] J. Barnes and P. Hut, "A hierarchical $\mathcal{O}(n \log n)$ force calculation algorithm," *Nature*, vol. 324, p. 446, 1986.

[6] W. Hackbusch, *Iterative Solution of Large Sparse Systems.* New York: Springer Verlag, 1994.

[7] W. Hackbusch, "A sparse matrix arithmetic based on $\mathcal{H}$-matrices—Part I: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 62, pp. 89–108, 1999.

[8] S. Kurz, O. Rain, and S. Rjasanow, "The adaptive cross-approximation technique for the 3D boundary-element method," *IEEE Trans. Magn.*, vol. 38, pp. 421–424, Mar. 2002.

[9] L. Grasedyck and S. Börm, private communication, 2002.