Massively Parallel Micromagnetic FEM Calculations with Graphical Processing Units (GPUs)

Elmar Westphal, Attila Kákay* and Riccardo Hertel*
IFF-Scientific IT-Systems, *IFF-9, Institut für Festkörperforschung, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany
e.westphal@fz-juelich.de

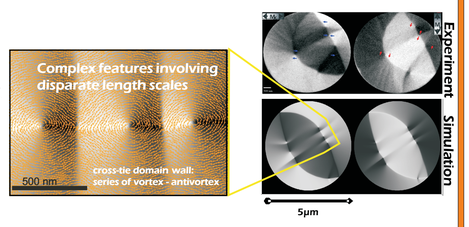
Introduction

Micromagnetic simulations is a powerful numerical tool for the study of magnetism in mesoscopic samples.

Complex magnetic structures - multiple length scales

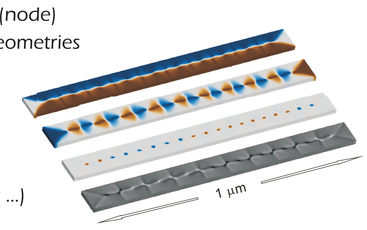
Micromagnetic structures can display a high complexity as a result of interacting length scale (vortices, domain walls, domains).

This complexity represents a challange for micromagnetic simulations.



What we do:

- Fully three-dimensional finite element modelling - FEM - (tetrahedral discretization cells)
- Calculate effective fields, energy densities etc. at each discretization point (node)
- Calculate the current-density distributions and Oersted fields in general geometries
- Solving Poisson's equation at each time step
- Integrate the Landau-Lifshitz-Gilbert differential equation
- Solve large sets of coupled integro-differential equations



What we obtain:

- Complex static and dynamic magnetic structures (e.g. vortices, spin waves ...)
- Solutions without any simplifying assumption on the magnetic structure
- Possibility of direct comparision with experimental results

FEM - BEM for magnetostatic fields with scalar potential

The magnetostatic field can be calculated as a gradient field of a scalar potential: $\vec{H}_s = -\vec{\nabla} U$

The scalar potential can be computed from: $U(\vec{x}) = \int_V \vec{M}(\vec{y}) \cdot \vec{\nabla} G(\vec{x},\vec{y}) d^3y$

$$\Delta U = \begin{cases} -\nabla \cdot \vec{M} & \text{inside the magnetic volume.} \\ 0 & \text{outside of the magnetic volume.} \end{cases}$$

The potential is split into two parts, $U = U_1 + U_2$

where U_1 is the solution of the inhomogeneous Neumann problem, while U_2 satisfies Laplace's equation with Dirichlet boundary conditions.

The calculation of the magnetostatic potential is done in three steps[1]:

1. An iterative solution of a sparse linear system for U_1. (linbcg)
2. A dense matrix or hierarchical matrix-vector multiplication to obtain the values of U_2 on the boundary of the magnetic region.
$$U_2^i = D_{ij} U_1^j$$
3. An iterative solution of sparse linear systems for U_2 within the magnetic region. (linbcg)

Equation of motion

Landau-Lifshitz-Gilbert equation:

$$\frac{d\vec{m}}{dt} = -\gamma \vec{m} \times \vec{H}_{eff} + \alpha \vec{m} \times \frac{d\vec{m}}{dt}$$

Effective field: $\vec{H}_{eff} = -\frac{1}{\mu_0 M_S} \cdot \frac{\partial e}{\partial \vec{m}}$

e: local energy density: Zeeman, exchange, magnetostatic and anisotropy energy density

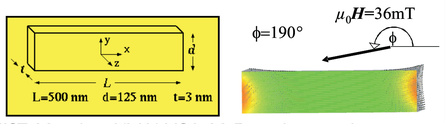
$$\vec{H}_{eff} = \frac{2A}{\mu_0 M_S} \Delta \vec{m} + \vec{H}_{ext} + \vec{H}_{stray} - \frac{1}{\mu_0 M} \frac{\partial e_K}{\partial \vec{m}}$$

Sample problem

Simulations have been performed on µMAG standard problem #4 with:

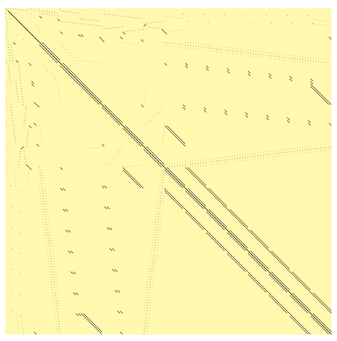
- various discretization - varying number of discretization nodes
- different complexity - regular and irregular mesh

Permalloy thin film element

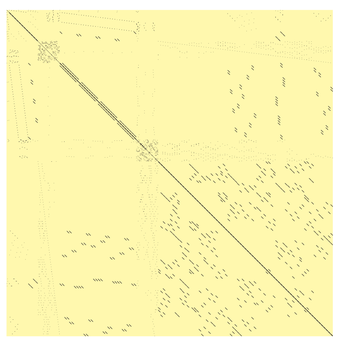


NIST, Maryland (VA) USA, M. Donahue et al.

The mesh type directly defines the structure of the sparse matrices and the performance of the calculations for both the CPU and GPU strongly depends on whether the mesh is regular or irregular.



For the regular mesh, the access to the vector elements during the matrix-vector multiplication follows certain patterns with strong data locality where many reads can be combined.



With the matrix generated from an irregular mesh, the access is seemingly random, a disadvantage especially for the GPU due to its small caches.

Test Setup

We benchmarked the simulation times on two different systems:

- CPU system: dual hex-core Intel X5670 2.93 GHz, 12 MB L3 cache
- GPU system: nVIDIA GTX480 in a Intel Core 2 quad 2.4 GHz based PC

- The speed-up for the solver is calculated from the average duration per solver iteration for the calculation of U1
- The speed-up for the ODE-integrator is calculated from the overall runtime of the integrator for 100ps of simulation time.
- All data (except the elements of the dense matrix) are stored and processed in double precision.
- Effective calculation speeds are calculated using the original matrix sizes, disregarding zero padding
- Since the main part of the calculations is performed by the GPU, multi-GPU systems can perform multiple jobs without significant speed loss. This was confirmed in different runs.

References

[1] D.R. Fredkin and T.R. Koehler, J. Appl. Phys. 63, 3385 (1988)
[2] S. Boerm and L. Grasedyck, HLib – A library for H - and H^2 - matrices, 1999, http://www.hlib.org/
[3] MMM2008, Austin, GS-04 High resolution large-scale micromagnetic simulations with hierarchical matrices, A. Kakay, S. Boerm and R. Hertel
[4] Numerical Recipes in C, http://www.nr.com/
[5] https://computation.llnl.gov/casc/sundials/main.html
[6] CUDA Programming Guide, http://www.nvidia.com
[7] N. Bell and M. Garland, "Efficient Sparse Matrix-Vector Muliplication on CUDA", NVIDIA Technical Report NVR-2008-004, December 2008
[8] Parallel reduction problems (norms and dot-products), http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf

Adaption of the TetraMag micromagnetic simulator to CUDA

The adaption of the existing code consists of three major parts:

Part 1: Solver (linbcg)

- Matrix is preprocessed for coalesced memory access
- Blocks of almost constant lenght (no need for Hybrid schemes [7])
- Reorder elements to optimize read-cycles/caching
- Overall flow control stays on CPU
- Zero-memcpy implementation for passing results
- Selection of fastest kernels based on problem structure
- Called for the calculation of U1 and U2

Part 2: Dense-Matrix-Vector Multiplication

- Performed on GPU for up to ~40k boundary nodes
- For more boundary nodes: hierarchical-matrix-vector multiplication on CPU (GPU-conversion: work in progress)
- Preprocessing similar to sparse matrices
- Lines are zero-packed and sorted by length to reduce zero-padding in blocks
- Called for the calculation of U2

Part 3: Time Integration

- Performed by CVODE from the Sundials Package [5]
- Includes field calculations as well as vector operations using the "NVector" structure supplied with Sundials
- "NVector" and field calculations were ported to CUDA
- Comparisons are done against an OpenMP enhanced version of the originally serial "NVector"

Preprocessing the Sparse Matrix:

1.: Original Sparse Matrix

	1	2	3	4	5	6
1	5		4			8
2		3			1	
3		2	7			4
4	9			2		
5			3		1	
6		2				6

2.: Sparse Matrix transposed

	1	2	3	4	5	6
1	5			9		
2		3	2			2
3	4		7		3	
4				2		
5		1			1	
6	8		4			6

3.: Sparse Matrix divided into warp-size-wide blocks (here warp-size=4 for demonstrative purposes)

1	2	3	4	5	6	0	0
5 (1)	3 (2)	7 (3)	2 (4)	1 (5)	6 (6)	0 (1)	0 (1)
4 (3)	1 (5)	2 (2)	9 (1)	3 (3)	2 (2)	0 (1)	0 (1)
8 (6)	0 (1)	4 (3)	0 (1)				

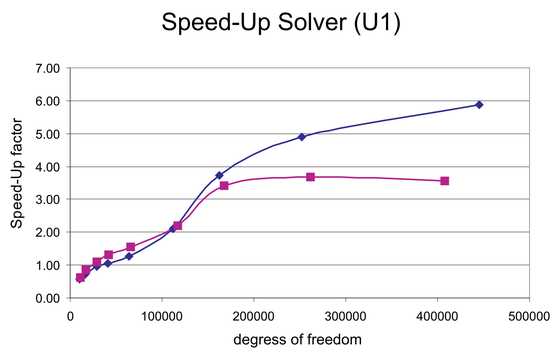
Zero-Padding

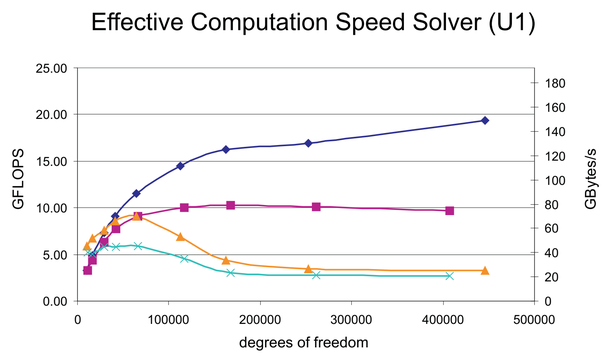
block start	0	12	20

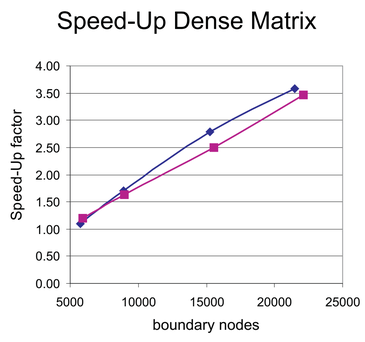
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
value	5	3	7	4	4	1	2	9	8	0	4	0	1	6	0	0	3	2	0	0
column	1	2	3	4	3	5	2	1	6	1	6	1	5	6	1	1	3	2	1	1

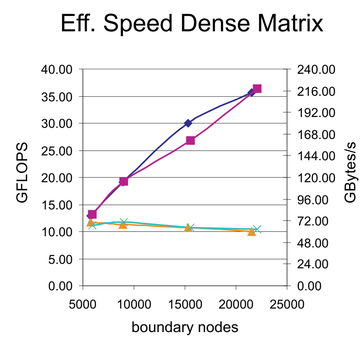
4.: Blocks are then stored sequentially in device memory and referenced. Multiple warp-sized blocks are combined to the thread-blocks of the actual operations. (note: reordering step not shown here)

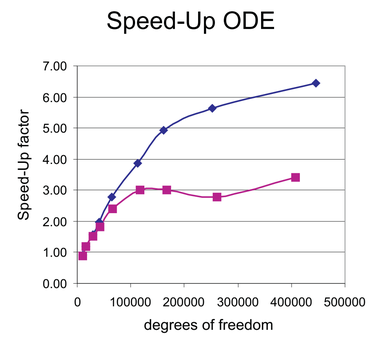
Benchmark results and comparison of calculation speeds: 1 GPU vs. 12 CPU cores

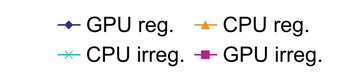










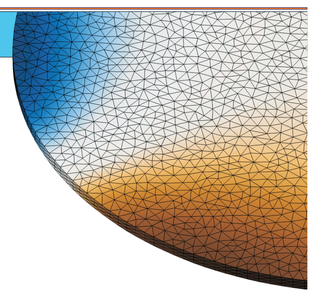


- The computation speed of the GTX480 equals or exceeds that of the 12-core setup already at relatively small problem sizes
- The initial advantage of the CPU's large caches wears off as the problem size outgrows it
- With increasing problem size, the larger overhead for the GPU calculations becomes less significant
- The GPU's speed advantage for regular problems increases for all problems that fit the available memory size
- For large problems, the solver operates near the theoretical memory bandwidth limit of the GPU
- Due to caching of the Fermi based architecure, the memory throughput of the dense-matrix-vector multiplication exceeds the posted bandwidth limit of the card

Conclusions

- GPU systems are very efficient for the Finite Element Micromagnetic simulations. The simulation time can be considerably reduced.
- The gain in computation speed generally increases with increasing numbers of discretization nodes. It exceeds the performance of the fastest available PC-based systems already at small problem sizes
- The GPU implementation allows us to systematically study the magnetization dynamics in large magnetic samples with reasonable computation time.
- Using more than one GPU per computer without performance penalty significantly increases cost efficiency.
- Bigger and faster device memory as well as bigger caches would be desirable to further utilize the GPUs high calculation throughput

TetraMag - finite element micromagnetic simulator



General-purpose finite-element algorithm to simulate the magnetization dynamics and the domains in ferromagnetic nanostructures.

TetraMag

- fully three-dimensional
- high geometric flexibility (hybrid FEM/BEM scheme)
- magnetostatic coupling (e.g., arrays of nanomagnets)
- easily portable (source code written in standard C)
- OpenMP parallelized
- ported to GPU architecture

Mitglied der Helmholtz-Gemeinschaft