Performance Improvement of Calculation of Static Magnetic Field of Micromagnetic Simulator Using Supercomputer FX10

Masahiro Arai
Department of Electrical Engineering and Electronics
Kogakuin University
Tokyo, 163-8677
Japan
cm17003@ns.kogakuin.ac.jp

Fumiko Akagi
Department of Applied Physics
Kogakuin University
Tokyo, 163-8677
Japan
fumiko.akagi@ns.kogakuin.ac.jp

Saneyasu Yamaguchi
Department of Information and Communications Engineering
Kogakuin University
Tokyo, 163-8677
Japan
sane@ns.kogakuin.ac.jp

Kazuetsu Yoshida
Kogakuin University
Tokyo, 163-8677
Japan
kyoshida@polka.plala.or.jp

ABSTRACT

A micromagnetic simulator is widely used for analyzing the dynamic behavior of magnetization. However, it is inefficient in terms of time consumption. In this paper, we investigated several methods for increasing the performance of the simulator using the supercomputer FX10. We show that parallelizing the first element of a data array makes it possible to speed up the calculation time for `MPI_Allgather()`. We also demonstrate that hybrid parallelization of the message passing interface (MPI) with `MPI_Allgather()` and OpenMP is effective.

CCS CONCEPTS

• **Computer systems organization~Embedded systems** • *Software and its engineering~Parallel programming languages* • Software and its engineering~Software notations and tools

KEYWORDS

Micromagnetic simulator, Fast Fourier transform (FFT), message passing interface (MPI), OpenMP, Hybrid MPI, and OpenMP parallelization

ACM Reference format:

M. Arai, F. Akagi, S. Yamaguchi, and K. Yoshida, 2017. SIG Proceedings Paper in Word Format. In *HPC Asia 2018, Chiyoda, Tokyo, Japan, January 2018*, 4 pages.

1 INTRODUCTION

A micromagnetic simulator is used for analyzing the dynamic behavior of magnetization, which shows the strength and the magnetization direction of a magnet, and is commonly used for designing various magnetic products including hard disk drives [1]. However, the calculation of static magnetic fields among a huge number of small nanosized cells, into which the simulation model discretizes the simulated field, is time consuming.

In this work, we used the supercomputer FX10 at the University of Tokyo to investigate the possibility of using parallelizing methods to decrease the calculation time. First, we present a parallelization of the message passing interface (MPI) with `MPI_Allgather()` for CPU-to-CPU communications with various directions of parallelization in its three-dimensional (3-D) model. Second, we compare three parallelizing methods for communication among CPUs, namely MPI [2], multithread parallelization with OpenMP [3], and OpenMP/MPI hybrid parallelization.

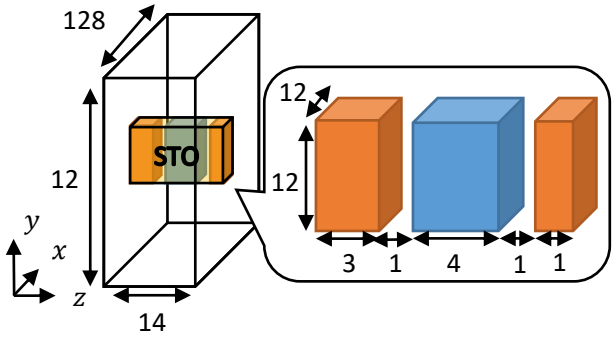

Figure 1: Simulation model of STO.

2 CALCULATION MODEL AND MICROMAGNETIC SIMULATOR

Fig. 1 illustrates the proposed simulation model for improving the performance. This shows the spin torque oscillator (STO), which is an element of the hard disk drive that makes it possible to achieve large recording capacity. In this case the STO contains three layers. When a magnetic field and a current are applied to the STO, the AC field generated from the STO is applied to the hard disk to assist with recording to the disk. The analysis field, including the STO and air, is divided into 2.5-nm rectangular prism cells. The number of cells in the X-, Y-, and Z-directions are 128, 128, and 14, respectively, which equates to 229,376 cells in total. This simulator uses C++ as the programming language and the differential equation as shown below is solved by Heun's method.

The dynamic behavior of the magnetization is calculated using a modified *Landau-Lifshitz-Gilbert* (LLG) equation with a spin torque field, as shown in equation (1).

$$(1+\alpha^2)\frac{d\vec{M}}{dt} = -\gamma\vec{M}\times\left(\vec{H}_{eff}-\alpha\vec{H}_{st}\right)-\frac{\gamma}{M_s}\vec{M}\times \left\{\vec{M}\times\left(\alpha\vec{H}_{eff}+\vec{H}_{st}\right)\right\}, \quad (1)$$

where $\boldsymbol{M}$ is the magnetization vector, t is the time, γ is the gyro magnetic constant, α is the damping constant, M_s is the saturation magnetization, and $\boldsymbol{H_{eff}}$ is the effective field vector. $\boldsymbol{H_{eff}}$ is composed of the static, anisotropy, external, and exchange magnetic fields. $\boldsymbol{H_{st}}$ is the spin-torque field applied to the STO via an electron spin. The calculation of the static magnetic field is quite time consuming. The static magnetic field is expressed as follows:

$$\begin{bmatrix}H_x\\H_y\\H_z\end{bmatrix} = \sum_{cell}\begin{bmatrix}S_{xx} & S_{xy} & S_{xz}\\S_{yx} & S_{yy} & S_{yz}\\S_{zx} & S_{zy} & S_{zz}\end{bmatrix}\cdot\begin{bmatrix}M_x\\M_y\\M_z\end{bmatrix}, \quad (2)$$

where $H_a(a{:}x,y,z)$ is the static magnetic field, S_{ab} $(b{:}x,y,z)$ is a structure factor depending on the shape of the cell and the cell-to-cell distance, and M_a is the magnetization. Thus, the static magnetic field is the sum of the static magnetic fields of all N cells including itself. Therefore, the required calculation time of the static magnetic fields at every time step is proportional to N^2, i.e., $O(N^2)$. This calculation time can be decreased to $O(N\cdot\log N)$ time, as shown in equation (3), by applying a fast Fourier transform (FFT).

$$H(k) = S(k)\cdot M(k), \quad (3)$$

where $H(k)$, $S(k)$, $M(k)$, and k are the static magnetic field, the structure factor, the magnetization, and the frequency, respectively. About 90% of the calculation time is occupied by the time required to calculate the static magnetic field as shown Fig. 2. Thus, decreasing this time is an important consideration.

Table 1 shows the specifications of a node in the supercomputer FX10. The number of available nodes is 12. The maximum number of processes in one node is 16. As a result, the number of available processes is 192. The maximum number of threads is also 16. Therefore, the product of the number of threads and the processes must be equal to or lower than 192.

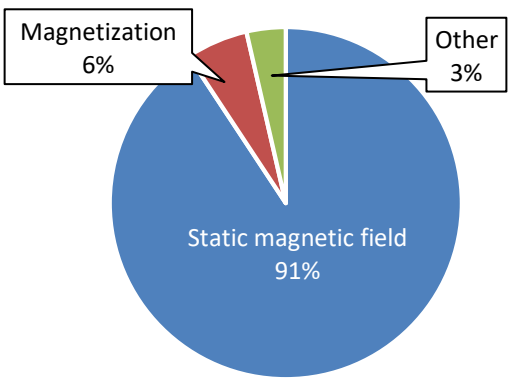

Figure 2: Breakdown of calculation time in LLG simulator.

Table 1: Specification of one node in FX10.

Processor	1.848 GHz,16core ×1
Theoretical peak performance	236.5 Gigaflops
Memory capacity	32 GB

3 PARALLELIZING METHODS

MPI was utilized by dividing the iterative processes and by inserting the codes for communication. This was achieved as follows. First, the number of CPUs and the rank are obtained by using an MPI API. Second, the iterative processes are equally divided into groups and each group is assigned to a process. Third, the calculation is performed in every process. Fourth, the calculation results of all the processes are integrated by the MPI communication function. In this work, we use `MPI_Allreduce()` and `MPI_Allgather()` as communication functions.

The second parallelization method we used was OpenMP, which is easier to use in practice because it only requires directives starting with `#pragma` to be inserted to affect the parallelization. Iterative processes are divided equally by inserting the directive of `#omp parallel for schedule(static) private()` before the `for` code of a loop.

Finally, we also investigated the use of hybrid parallelization, which is a method that utilizes both MPI and OpenMP.

The micromagnetic simulator has two triple `for` loops, which are used for the FFTs of the static magnetic fields or the magnetization. Two of the loops in the FFT are in the X- and Y-directions, respectively. The triple loops contain loops in the X-, Y-, and Z-directions, and these can be parallelized. Furthermore, when the subscript of the 3-D array is [i][j][k], the elements of [i], [j], and [k] are termed the first, second and third elements, respectively.

4 PERFORMANCE EVALUATION AND DISCUSSIONS

4.1 Parallelization of MPI with `MPI_Allgather()`

We discuss the relationship between the number of processes and the calculation time in `MPI_Allgather()`. The relationship is shown in Fig. 3. Comparing the parallelization of the `for` loop in the Z-direction and those in the X-and Y-directions, we can see that the parallelization in the Z-direction is faster than those in the latter two directions. This simulator applies two-dimensional FFT. The Fourier transforms in the X- and Y-directions are sequentially performed. However, if the data in the direction of the FFT in the triple loop were parallelized, the calculation result would be incorrect. Therefore, in the case of the FFT in the X-direction, the data in the Y-direction should be parallelized and integrated by `MPI_Allgather()`. The FFT in the Y-direction is achieved by the same method. On the other hand, in terms of parallelization in the Z-direction, it is unnecessary to perform `MPI_Allgather()` after the FFT in the X-direction, hence the communication time is shortened. As a result, parallelization in the Z-direction is faster than in the X- and Y-directions.

Moreover, parallelization in the X-and Y-directions means that the second and third elements of the 3-D array [Z][Y][Z] are paralellized. When non-first elements are parallelized and integrated by `MPI_Allgather()`, the integrated data are transposed as shown in Fig. 4. Therefore, it is necessary to re-order the integrated data after `MPI_Allgather()`. In this study, we temporarily store the integrated data in another data array, and then copy the data to the original data array with re-ordering. These procedures should be omitted to reduce the time consumption.

For that purpose, it is needed to parallelize the first element of the 3-D array. Then, the data are integrated in the appropriate order by `MPI_Allgather()`. For example, the elements should be re-arranged in the order of [Y][Z][X] by transposing the array when the FFT is performed in the X-direction. Fig. 5 compares the conventional and proposed methods by providing a breakdown of the execution time for the 16 processes. This result indicates that the improved method is 20% faster than the conventional one because of the reduction in the number of times the data needs to be re-ordered.

4.2 Comparison of parallelizing methods

In this section, we compare the parallelizing methods. First, we compare `MPI_Allreduce()` and `MPI_Allgather()` for MPI. As shown in Fig. 6, `MPI_Allgather()` outperformed `MPI_Allreduce()`. This is mainly because the amount of transmitted data of `MPI_Allgather()` are less than those of `MPI_Allreduce()`. The maximum speed of the parallelization in the Z- direction with `MPI_Allgather()` is approximately 11.8 times higher compared to the speed with one processor.

Next, we focus on hybrid parallelization. Fig. 7 shows the relationship between the calculation time and the number of threads and processes when using hybrid parallelization. The results show that the performance was improved by 20.9 times and 22.0 times at most with 7 and 14 processes, respectively. Comparing the results in Figs. 6 and 7, we can see that the hybrid method achieved higher performance.

The above results enable us to conclude that hybrid parallelization is more suitable than parallelization with only MPI for the LLG simulator. In addition, we can also conclude that `MPI_Allgather()` is more effective than `MPI_Allreduce()`.

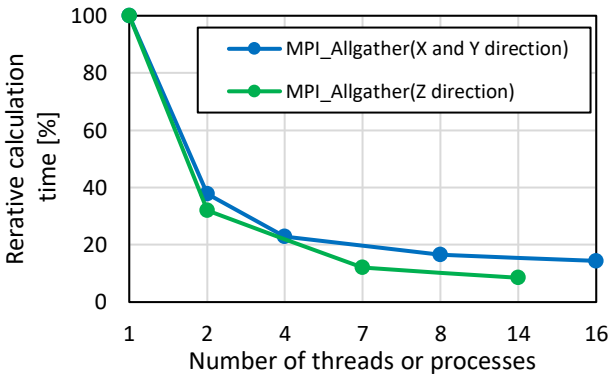

Figure 3: Relationship between number of processes and calculation time in `MPI_Allgather()`.

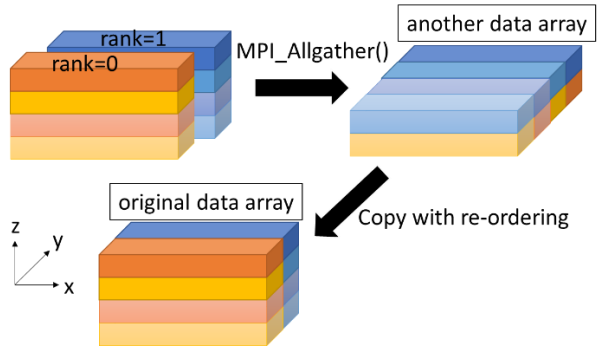

Figure 4: Schematic of data array for MPI parallelization in the X- and Y-directions using `MPI_Allgather()`.

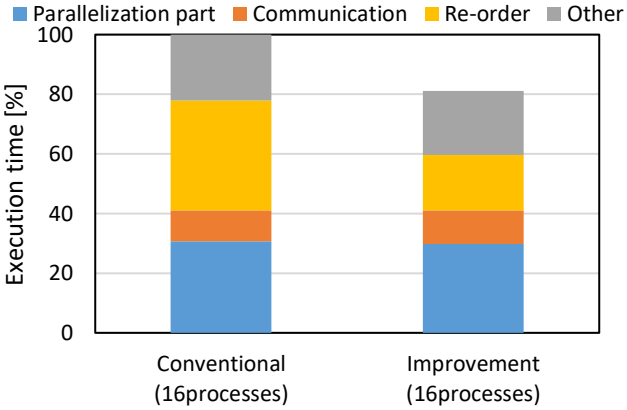

Figure 5: Comparison between conventional and improved methods in breakdown of execution time for `MPI_Allgather()`.

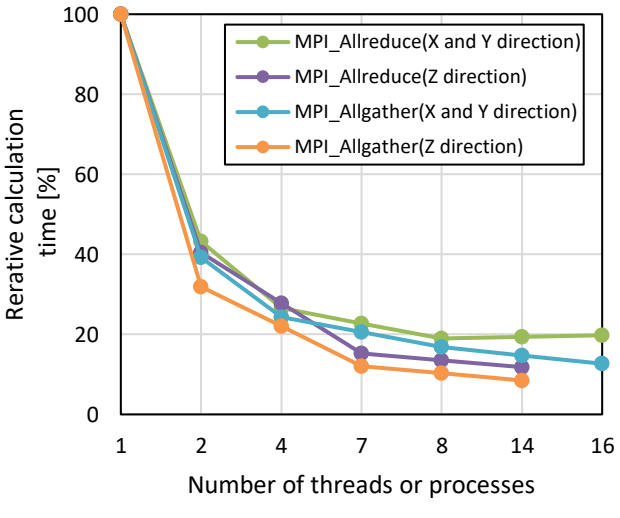

Figure 6: Relationship between number of processes and calculation time using MPI.

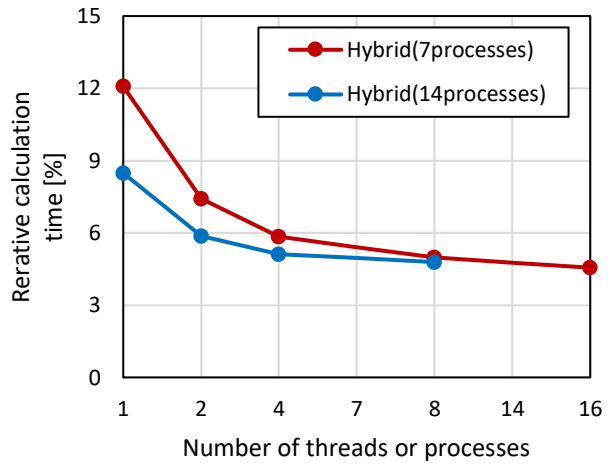

Figure 7: Relationship between number of threads and calculation time using hybrid parallelization.

5 CONCLUSIONS

In this study, we investigated several parallelizing methods using the supercomputer FX10 at the University of Tokyo. In the case of parallelization of MPI with `MPI_Allgather()`, the parallelization in the Z-direction is faster than those in the X- and Y-directions, because using `MPI_Allgather()` after the parallelization in the X-direction can be omitted. In addition, the calculation time can be decreased by parallelizing the first element of the data array. Our evaluation demonstrated that the calculation time can be reduced by applying hybrid parallelization using MPI with `MPI_Allgather()` in the Z-direction (without FFT) and OpenMP in the X- and Y-directions (with FFT for the static magnetic field and magnetization).

ACKNOWLEDGMENTS

This study was supported in part by the Storage Research Consortium Japan. This work was supported by JST CREST Grant Number JPMJCR1503, Japan. This work was supported by JSPS KAKENHI Grant Numbers 26730040, 15H02696, 17K00109.

REFERENCES

[1] Y. Tang, and J. G Zhu, *IEEE Trans. Magn*. 44, 11 (2008) 3376-3379 (2008).
[2] P. Pacheco, H. Akiba. 2001. *Parallel Programming with MPI* (in Japanese), Baifukan, Tokyo, 43-56.
[3] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon. 2001. *Parallel Programming in OpenMP*, Morgan Kaufmann.