The Parallelized Automatic Mesh Generation Using Dynamic Bubble System with GPGPU

Fumiaki Nobuyama, So Noguchi, and Hajime Igarashi

Graduate School of Information Science and Technology, Hokkaido University, 060-0814 Sapporo, Japan

An automatic mesh generation method employing a dynamic bubble system can provide a high-quality mesh for electromagnetic finite element analysis. However, there is a problem that it takes very long time to compute bubbles' movement when a mesh with a huge number of elements is produced. It is possible to independently and simultaneously compute the bubbles' movement, therefore the computation of the bubbles' movement is suitable for a parallel computing. In order to shorten the computation time, the computation of bubbles' movement is parallelized with GPU. We propose a dynamic bubble system parallelized with GPU. As the result, the reduction of the computation time was achieved.

***Index Terms*—Dynamic bubble system, finite element analysis, mesh generation, parallel computing.**

I. Introduction

RECENTLY, Finite Element Analysis (FEA) is frequently done for design and performance survey of electromagnetic machines. As a preprocessing of FEA, mesh generation is necessarily required. It is well known that a mesh of good quality yields high analysis accuracy and short computation time. However, the mesh generation is usually very laborious and time consuming, especially in 3-D space. In order to solve the problem, a large number of methods for automatic mesh generation have been proposed. Among them, a 3-D automatic mesh generation method adopting a dynamic bubble system [1]-[3] is a promising mesh generation method. It is constructed with a dynamic bubble system and Delaunay division [4]. The dynamic bubble system generates a set of vertices inside an entire analysis region, and then the Delaunay division completes connection of the vertices to make a mesh.

The 3-D automatic mesh generation method employing the dynamic bubble system, proposed in [1], has a high ability to generate a high-quality mesh. On the other hand, it has a problem that it needs long computation time. That is, it requires a large computational cost to simulate bubbles' movement, like an *N*-body problem. In the bubbles' movement simulation, numerous bubbles move independently and simultaneously. Consequently, it is easily possible to parallelize the computation of the bubbles' movement in order to shorten the computation time of mesh generation. Accordingly, we improve the 3-D automatic mesh generation method [1] by employing a Graphics Processing Unit (GPU) as a hardware device for the purpose of a parallel computing.

General-Purpose Computing on GPU (GPGPU) is a hardware acceleration device and recently has more than 500 cores. The GPGPU shows so high performance of a parallel computing that it can shorten the computation time of the bubbles' movement mentioned above. So far, a few solvers with GPGPU for FEA have been proposed [5]-[8], however a method of mesh generation with GPU has not been reported yet.

Generally OpenMP or MPI is also used for parallel computing. However, since computation complexity on the movement of each bubble is very low and the number of bubbles is very large, the GPGPU is more suitable for the parallelized mesh generation using the dynamic bubble system than the OpenMP and the MPI.

In this paper, we propose a method to parallelize the dynamic bubble system with GPGPU and aim to accelerate an automatic tetrahedral mesh generation with keeping the quality of mesh.

II. Dynamic Bubble System

A. Dynamic Bubble System

The dynamic bubble system [1] is a physical model using multiple bubbles. It generates and moves many bubbles in an analysis domain, and computes a dense disposition of them as possible. The disposition of nodes obtained by the dynamic bubble system has a smooth variation of sparse and dense. Therefore, it is promised that a high-quality mesh is generated [9], [10].

The bubble is usually a spherical particle which has a node at the center point, a radius, mass, and volume. The center location and radius of bubbles determine the bubbles' motion with their interaction in the analysis domain, wherein the bubbles ultimately reach their stabilized disposition. The radius of each bubble is proportional to a desired mesh size, and each bubble acts on a force from other bubbles. Fig. 1 shows the procedure of the ordinary dynamic bubble system [1].

The ordinary dynamic bubble system has a problem that long computation time is necessary. Let the number of bubbles be N, and $N \times (N-1)$ computations is necessary in every time step. Consequently, the computation time explosively increases with increase of bubbles. However, recently, large-scale FEAs with several hundred thousand elements have been easily and frequently done, therefore the dynamic bubble system has to be improved to deal with several hundred thousand elements at least with short time as possible.

Manuscript received October 30, 2012. Corresponding author: S. Noguchi (e-mail: noguchi@ssi.ist.hokudai.ac.jp).

Digital Object Identifier inserted by IEEE

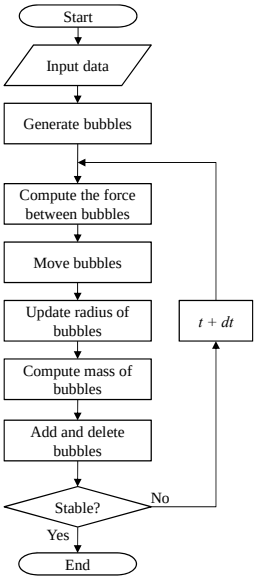


Fig. 1. Flowchart of the ordinary dynamic bubble system.

B. Parameters of bubble motion

The following equation of motion is considered in the dynamic bubble system.

$$m_i \frac{\mathrm{d}^2 p_i}{\mathrm{d}t^2} + c_i \frac{\mathrm{d}p_i}{\mathrm{d}t} = f_i \quad (i = 1,2,\ldots,N) \tag{1}$$

where m_i is the mass of bubble i, c_i is the viscosity coefficient of bubble i, p_i is the position of the bubble center, N is the total number of bubbles, and f_i is the resultant force acting on bubble i, respectively. The force f_i, which depends on the center position and the distances from its center to the center of the neighboring bubbles, is mathematically modeled by the van der Waals force. Let d denote the distance between the centers of two adjacent bubbles, and the van der Waals force acting on the bubbles is approximated by the 3rd order polynomials, as shown in Fig. 2. The approximation function is set to the following equations.

$$f(d) = \begin{cases} Ad^3 + Bd^2 + Cd + D, & 0 \le d \le d_1 \\ 0, & d < 0,\ d_1 < d \end{cases} \tag{2a}$$

$$f(d_0) = 0 \quad f(d_1) = 0 \quad f'(0) = 0 \quad f'(d_0) = -k_0 \tag{2b}$$

where $f(d)$ indicates the magnitude of the force between two neighboring bubbles and k_0 is the elastic coefficient at the stabilized distance r_0. Note that the force at $d = 0$ is not infinite in this model, and the forces are activated within a limited distance ($d < d_1 = 1.5\ d_0$). The finite force at $d = 0$ prevents a singular situation when the center of bubbles coincide with each other. To reduce the computation cost, only the bubbles in a limited distance ($d < d_1$) are considered in the dynamic bubble system. k_0 is the absolute value of the differential coefficient of the van der Waals' force and is expressed as follows

$$k_0 = \frac{72}{d_0^2}. \tag{3}$$

The initial forces $f_i(d)$ are defined from the disposition of the initial bubbles, and the final position should be determined by enforcing the system of (1) by iteration until a stable state is achieved.

The bubbles move until they reach the stable state. A filling ratio Q is used as a stability criterion. Q is defined as follows

$$Q = \frac{V_{\text{bubble}}}{V_{\text{region}}} \tag{4}$$

where V_{bubble} is the volume of total bubbles and V_{region} is the volume of the analysis region, respectively. V_{bubble} is expressed as follows

$$V_{\text{bubble}} = \sum_{i=1}^{N} \alpha \left(V_i - \sum_{j=i+1}^{N} V_{ij} \right) \tag{5}$$

where V_{ij} is the overlap volume of bubbles i and j, N is the total number of bubbles and α is the volume adjustment coefficient, respectively.

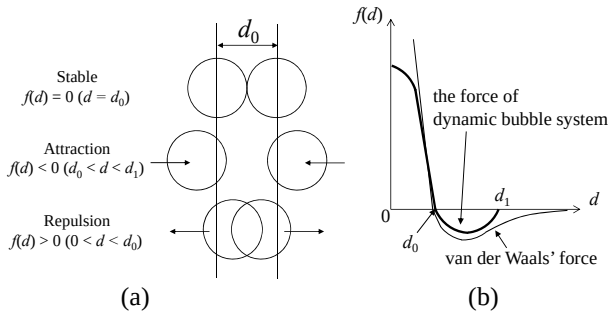


Fig. 2. Interbubble proximity-based forces. (a) Change of force depending on distance between two bubbles, and (b) simplified approximate force and exact van der Waals force.

III. Bubble System Parallelized with GPGPU

The calculation of a force between bubbles needs an N-loop computation when there are N bubbles in an analysis region. On a CPU, the computation is performed in turn from 1 to N. On the other hand, a GPU can perform the computation in parallel as shown in Fig. 3.

The calculation of the force between bubbles is suitable for a parallel computing, and it is easily implemented with CUDA [11] for GPGPU. However, when only the force computation is parallelized with GPGPU, it takes very long time to communicate bubble data between a main memory for CPU and a device memory for GPU. Therefore, the amount of the data communicated between the main and the device memory has to be reduced in order to achieve to shorten the computation time. In this paper, not only the computation of the force between bubbles but also parallelizable processes are parallelized. Fig. 4 shows the flowchart of the proposed parallelized dynamic bubble system. A process concerned with a few bubbles, that is the "add and delete bubbles" process, is unsuitable for parallelization. Conversely, the other processes concerned with almost all the bubbles are exceedingly suitable for parallelization, so that all the processes except the "add and delete bubbles" process are parallelized, as shown in Fig. 4. Accordingly, it is very important to decide what process is parallelized or not, because it takes very long time to communicate data between

the main memory for CPU and the device memory for GPU.

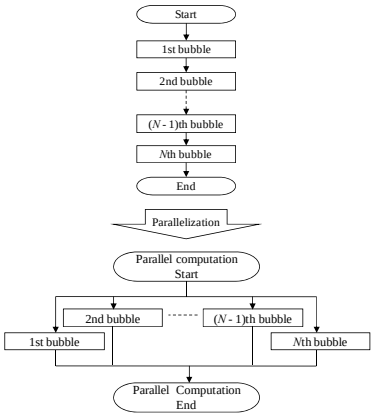


Fig. 3. Parallelization of loop statement with GPGPU.

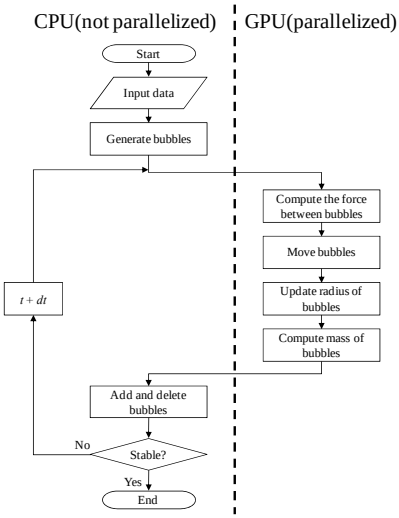


Fig. 4. Flowchart of the parallelized dynamic bubble system.

IV. Comparison of Computation Time in CPU and GPU

The usefulness of the automatic mesh generation employing the dynamic bubble system parallelized with GPGPU is verified on an example. An iron core model consisting of an iron core and a coil in 3-D space was meshed with a variety of number of elements. Fig. 5 shows the schematic drawing of the iron core model, and Fig. 6 shows one of meshes which were generated by the dynamic bubble system parallelized with GPGPU. In the GPU computation, the number of the threads which were launched on GPU was the number of the bubbles divided by 16.

Table I and Fig. 7 show the comparison of the computation time with a GPU (NVIDIA Tesla C2050) and with only a CPU (Intel Core i7 950 QuadCore 3.06 GHz). Here, a reduction rate γ defined as follows is used as an indicator of acceleration.

$$\gamma = \frac{t_{\mathrm{CPU}} - t_{\mathrm{GPU}}}{t_{\mathrm{CPU}}} \times 100 \qquad (6)$$

where t_{CPU} and t_{GPU} are the computation time in the case of CPU and GPU, respectively. The speed-up ratio s is also defined as $(t_{\mathrm{CPU}}) / (t_{\mathrm{GPU}})$.

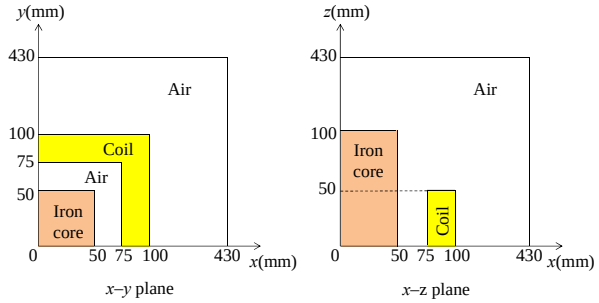
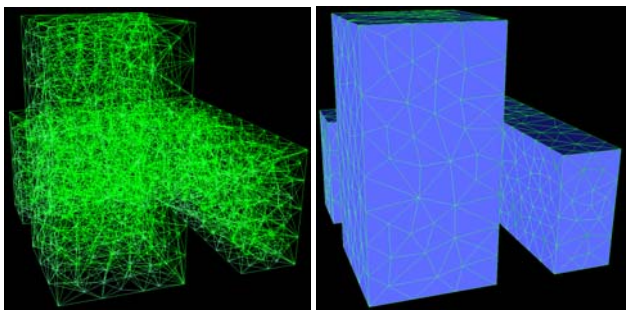


Fig. 5. Iron core model consisting of an iron core and coil in 3-D space.



(a) (b)

Fig. 6. Mesh of iron core model. It is generated by the parallelized dynamic bubble system. (a) General view, and (b) outside surface view. The mesh of air region is not depicted.

In the case of small number of bubbles, a high rate of computation time reduction could not be attained. However, as the number of bubbles increases, the reduction rate gradually rises. It is achieved to drastically shorten the computation time of the dynamic bubble system by parallelizing with GPGPU when a sufficiently large number of elements are dealt with.

TABLE I
Computation Time and Decreasing Rate

Number of bubbles	Computation time of CPU [min.]	Computation time of GPU [min.]	Decreasing rate [%]	Speed-Up ratio
10,000	0.23	0.17	25.9	1.4
20,000	0.63	0.40	36.8	1.6
50,000	10.5	3.7	65.0	2.9
100,000	32.7	10.8	66.9	3.0
200,000	82.8	24.1	70.9	3.4

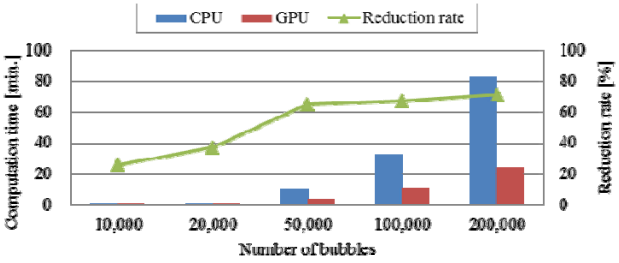


Fig. 7. Comparison of computation times with GPU and CPU.

Table II and Fig. 8 show the comparison of the computation time of each process in the case of 10,000 bubbles. Similarly, Table III and Fig. 9 show the comparison in the case of 200,000 bubbles. In both two cases, a high parallelization

efficiency of the “compute the force between bubbles” and the “update information of bubbles” process is achieved, because these processes require an identical computation to all the bubbles. On the contrary, the reduction rate of the “calculate packing ratio” process is negative, because this process needs a different amount of computation to all the bubbles according to (5). Therefore, this process is unsuitable for a parallel computing when the number of bubbles is not so large. However, the “calculate packing ratio” process is performed in GPU in order to reduce the amount of data transferred between CPU and GPU.

The “other” process contains the rest of all the processes, such as the data transfer between CPU and GPU, the memory initialization, and so on, but all the contents of the “other” process in the computation with GPU do not accord with those with only CPU. The reason why the reduction rate of the “others” process also shows a negative value is that the GPU has to transfer data to the main memory for CPU. Data such as bubble arrangement information cannot be shared between the CPU and the GPU. The data has to be transferred at every step from CPU to GPU and from CPU to GPU. The transfer time is a hindrance to shorting computation time. On the other hand, the data transfer between CPU and GPU is unnecessary in the mesh generation with only CPU.

When the number of bubbles is large enough, it is achieved to shorten the computation time of all the process except the “others” process. Moreover, the ratio of the data transfer time to the time of the other process becomes small so that high parallelization efficiency is achieved.

V. Conclusions

In this paper, we propose an automatic mesh generation adopting a dynamic bubble system parallelized with GPGPU, and the acceleration of automatic mesh generation using a dynamic bubble system was achieved with the GPGPU. Since a few processes unsuitable for a parallel computing take very long time, a parallelization effect notably appears in case of a huge number of bubbles. We have achieved 70.9% reduction of the computation time by parallelizing with GPGPU.

References

[1] T. Yokoyama, V. Cingoski, K. Kaneda, and H. Yamashita, “3-D Automatic Mesh Generation for FEA Using Dynamic Bubble System,” *IEEE Trans. Magn.*, vol. 35, no. 3, pp. 1318-1321, May 1999.

[2] S. Nagakura, S. Noguchi, K. Kaneda, H. Yamashita, and V. Cingoski, “Automatic quadrilateral mesh generation for FEM using dynamic bubble system,” *IEEE Trans. Magn*., vol. 37, no. 5, pp. 3522-3525, May 2001.

[3] Y. Marechal, D. Armand, and D. Ladas, “An adaptive remeshing technique ensuring high quality meshes,” *IEEE Trans. Magn*., vol. 44, no. 6, pp. 1222-1225, Jun. 2008.

[4] D. N. Shenton and Z. J. Cendes, "Three-Dimensional Finite Element Mesh Generation Using Delaunay Tessellation", *IEEE Trans. Magn.*, vol. 21, no. 6, pp. 2535-2538, Nov. 1985.

[5] N. Gödel, N. Numn, T. Warburton, and M. Clemens, “Scalability of Higher-Order Discontinuous Galerkin FEM Computations for Solving Electromagnetic Wave Propagation Problems on GPU Clusters,” *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 3469-3472, Aug. 2010.

[6] N. Gödel, N. Numn, T. Warburton, and M. Clemens, “GPU Accelerated Adams-Bashforth Multirate Discontinuous Galerkin FEM Simulation of High-Frequency Electromagnetic Fields,” *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 2375-2738, Aug. 2010.

[7] Y. Liu, S. Jiao, W. Wu, and S. De, “GPU Accelerated Fast FEM Deformation Simulation,” *APCCAS*, pp. 606-609, 2008.

[8] S. Ikuno, Y. Kawaguchi, N. Fujita, T. Itoh, S. Nakata, and K. Watanabe, “Iterative solver for linear system obtained by edge element: variable preconditioned method with mixed precision on GPU,” *IEEE Trans. Magn*., vol. 48, no. 2, pp. 467-470, Feb. 2012.

[9] K. Shimada and D. C. Gossard, “Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing,” *ACM 3rd Symposium on Solid Modeling and Applications,* pp. 409-419, 1995

[10] J. H. Kim, H. G. Kim, B. C. Lee, and S. Im, “Adaptive mesh generation by bubble packing method,” *Structural Engineering and Mechanics*, vol. 15, no. 1, pp. 135-149, 2003.

[11] NVIDIA, “NVIDIA CUDA Programming Guide v1.1,” http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf, Nov. 2007.

TABLE II
COMPUTATION TIME AND REDUCTION RATE OF 10,000 BUBBLES

Number of bubbles	Computation time of CPU [sec.]	Computation time of GPU [sec.]	Reduction rate [%]	Speed-Up ratio
Compute the force between bubbles	6.5	1.5×10^{-4}	100.0	4.2×10^{4}
Updating information of bubbles	1.6	9.2×10^{-4}	99.9	1.8×10^{3}
Add and delete bubbles	6.8×10^{-4}	2.4×10^{-4}	64.8	2.8
Calculate packing ratio	3.9	7.1	-81.7	0.6
Others	1.5	2.9	-99.4	0.5
Total	13.5	10.0	25.9	1.4

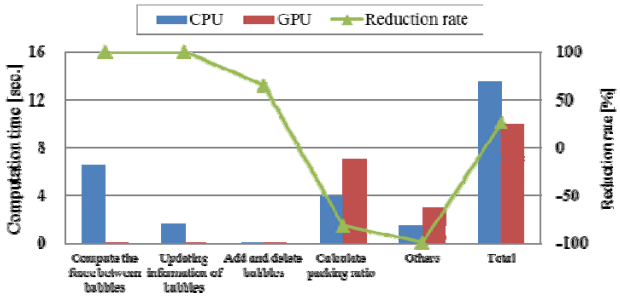


Fig. 8. Comparison of computation times of each process with GPU and CPU in the case of 10,000 bubbles.

TABLE III
COMPUTATION TIME AND REDUCTION RATE OF 200,000 BUBBLES

Number of bubbles	Computation time of CPU [min.]	Computation time of GPU [min.]	Reduction rate [%]	Speed-Up ratio
Compute the force between bubbles	44.1	1.6×10^{-6}	100.0	2.8×10^{7}
Updating information of bubbles	11.0	9.8×10^{-6}	100.0	1.1×10^{6}
Add and delete bubbles	1.3×10^{-5}	2.1×10^{-6}	83.4	6.0
Calculate packing ratio	27.0	22.6	16.2	1.2
Others	8.3×10^{-1}	1.4	-71.4	0.6
Total	83.0	24.1	71.0	3.5

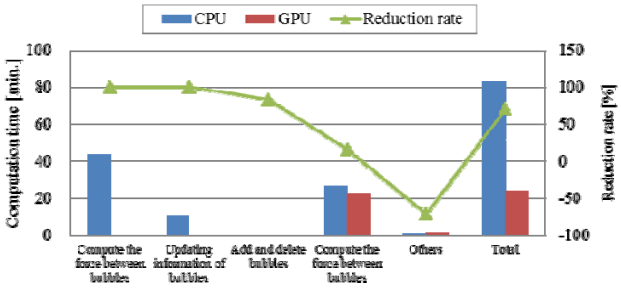


Fig. 9. Comparison of computation times of each process with GPU and CPU in the case of 200,000 bubbles.