Parallel Implementation and Branch Optimization of EBE-FEM Based on CUDA Platform

Yan Zhang, Xiuke Yan, Xudong Ren, Sheng Wang, Dongyang Wu, and Baodong Bai

School of Electrical Engineering
Shenyang University of Technology, Shenyang, 110870, China
yanxke@126.com

Abstract ─ The finite element analysis of large complex structures makes higher demand on memory capacity and computation speed, which leads to the inefficiency of traditional serial finite element method (FEM) for such large-scale problems. In this paper, the element-by-element finite element method (EBE-FEM) has been implemented parallelly on CUDA (Compute Unified Device Architecture) platform, and been programmed using C++ language. The thread branches that exist in parallel reduction program have been researched and optimized to improve parallel efficiency. The correctness of algorithms and programs are verified by the analysis of an open slot of motor. The optimized parallel program is applied to analyze the main magnetic field of a single-phase transformer. The results show that the EBE-FEM implemented on CUDA platform is more effective than serial EBE-FEM, and branch optimization can improve the speedup further.

Index Terms ─ Branch optimization, CUDA, EBE-FEM, parallel computation.

I. INTRODUCTION

When FEM is used to analyze the electromagnetic field of large electrical equipment, huge amounts of meshes are needed to get more accurate results, which leads to a large scale of computation. Using traditional serial FEM to calculate large-scale numerical problems, there will be problems such as too long calculation time and large calculation error, sometimes even unable to calculate. Therefore, more and more parallel FEM (PFEM) are developed and applied to numerical calculation and EBE-FEM is one of them [1]. EBE-FEM avoids the formation and storage of the global coefficient matrix, and has no requirement or restriction on the geometric shape and element number of the structure in the region [2,3]. The parallel computing of EBE-FEM can be implemented on the elements level, it has high parallelism. Combining EBE-FEM with advanced computing platform can solve the bottleneck problem of large-scale numerical computation.

GPUs represent one of the newest types of parallel processors. For its many-core nature, it is widely used in parallel calculation [4,5]. CUDA is a CPU+GPU heterogeneous parallel computing platform [6,7], which provides a reliable programming environment for GPU to perform data parallel processing. The EBE-FEM can be thought as a method which transforms a highly memory dependent problem to a massively computational dependent one, the latter can be parallelized efficiently. CUDA platform can make full use of the advantages of high parallelism of EBE-FEM [8,9]. GPUs are very good at computing, but weaker in logical judgments. For high-parallel and intensive computing, whether it is a small problem or a large problem, running on GPU in parallel is better than it on CPU in serial. Recent years, some CUDA function libraries, such as cuBLAS, cuBLAS-XT, etc. have been developed to implement computation on GPUs. However, they sometimes show poor speedup performance due to their non-optimized operating process. Thread branch may appear in the process of EBE-FEM parallel implementing on CUDA platform. It is one of the main factors to reduce the parallel efficiency. This problem can be solved by thread-data remapping method [10].

In this paper, EBE-FEM combined with Jacobi preconditioned conjugate gradient (J-PCG) method has been researched to realize parallel computation on CUDA platform. The thread branches in reduction operation has been investigated and thread-data remapping method on instruction-level is proposed to optimize addressing process of reduction operation. All the corresponding programs have been developed in C++. The correctness of the proposed algorithms and programs are verified by the analysis of an open slot of motor. The algorithms and programs have been applied to calculate the magnetic field in a single phase power transformer, the results have been analyzed and discussed.

II. METHOD DESCRIPTION

A. EBE technique

Using FEM, the Maxwell electromagnetic field equations can be discretized as linear equations:

Submitted On: December 17, 2019
Accepted On: April 12, 2020

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

where $\mathbf{A}$ is the whole stiffness matrix, $\mathbf{x}$ the vector of unknown variables and $\mathbf{b}$ the system vector.

According to EBE-FEM, (1) can be expressed as:

$$\left(\sum_{e=1}^{E}\left(\mathbf{Q}^{(e)}\right)^T \mathbf{A}^{(e)}\mathbf{Q}^{(e)}\right)\left(\sum_{e=1}^{E}\left(\mathbf{Q}^{(e)}\right)^T \mathbf{x}^{(e)}\right)=\left(\sum_{e=1}^{E}\left(\mathbf{Q}^{(e)}\right)^T \mathbf{b}^{(e)}\right), \tag{2}$$

where $\mathbf{A}^{(e)}$ is element stiffness matrix, $\mathbf{b}^{(e)}$ the element vector and matrix $\mathbf{Q}^{(e)}$ represents transition between local and global numbering of the unknown variables for the element.

The main operation of CG method is the inner product of vectors, which is appropriate to realize parallel computation. Therefore, CG method has been used to solve the equations of EBE-FEM. The iteration of CG method can be expressed by:

$$\mathbf{x}_0 = \left(0, 0, ...0\right)^T, \mathbf{r}_0 = \mathbf{P}_0 = \mathbf{b}, \tag{3}$$

$$\alpha_k = \frac{\left(\mathbf{P}_k, \mathbf{b}\right)}{\left(\mathbf{P}_k, \mathbf{AP}_k\right)}, \tag{4}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{P}_k, \tag{5}$$

$$\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{Ax}_{k+1}, \tag{6}$$

if $\mathbf{r}_k < \varepsilon$, stop iteration, $\mathbf{x} = \mathbf{x}_k$; else, update β_k and $\mathbf{P}_k$:

$$\beta_k = \frac{\left(\mathbf{r}_{k+1}, \mathbf{AP}_k\right)}{\left(\mathbf{P}_k, \mathbf{AP}_k\right)}, \tag{7}$$

$$\mathbf{P}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{P}_k, \tag{8}$$

return to the equation (4), and continue iteration process.

In EBE method, the inner product (**r, r**) and (**P**, **AP**) can be calculated on each element as follow:

$$(\mathbf{r}, \mathbf{r}) = \mathbf{r}^T\mathbf{r} = (\mathbf{Q}^T\mathbf{r}^e)^T\mathbf{Q}^T\mathbf{r}^e = \sum_{e=1}^{E}(\mathbf{r}^e)^T\mathbf{s}^{(e)}, \tag{9}$$

$$\mathbf{s}^{(e)} = \mathbf{r}^e \oplus \sum_{j=adj(e)} \mathbf{r}^j, \tag{10}$$

$$\left(\mathbf{P}, \mathbf{AP}\right) = \left(\mathbf{QP}\right)^T \mathbf{A}^e\mathbf{QP} = \sum_{e=1}^{E}\left(\mathbf{P}^{(e)}\right)^T \mathbf{A}^e\mathbf{P}^{(e)}, \tag{11}$$

$$\mathbf{Q} = \left(\mathbf{Q}^{(1)T}, \mathbf{Q}^{(2)T}, ..., \mathbf{Q}^{(E)T}\right)^T, \tag{12}$$

where $\mathbf{r}$ is the global residual vector, $\mathbf{r}^e$ is the local element residual vector, $\oplus$ refers to accumulation of the contribution made by all elements to the nodes of the element and adj(e) represents the adjacent element which share the common node with element e.

In order to improve its convergence further, the Jacobi preconditioned (JP) technology is applied to EBE-CG. The mathematic model of EBE-J-PCG was presented in [9]. The multi-core nature of GPU can exploit the parallelism of EBE-CG method considerably on CUDA platform.

B. Parallel realization on CUDA platform

CUDA is CPU+GPU heterogeneous computing platform, the programming model ensures that the GPU and CPU complement each other, which executes the complex logic control tasks on CPU and data-parallel computation-intensive tasks on GPU. Implementing EBE-FEM on CUDA, the CPU+GPU collaborative computing model is shown in Fig. 1.

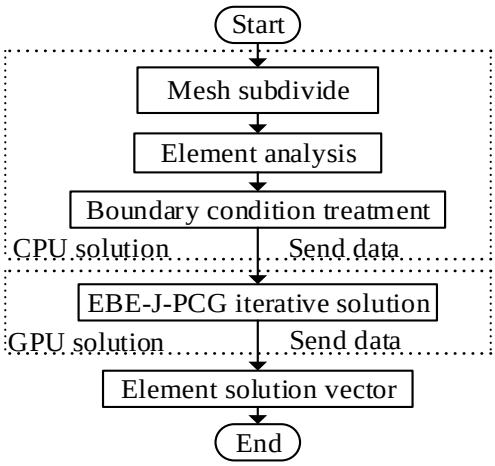

Fig. 1. Flow of CPU+GPU heterogeneous computing model of EBE-FEM.

In GPU solution, the EBE-J-PCG iterating solution for all elements can be operated at the same time. The calculations are performed by kernels with different function on GPU. The parallel computation on CUDA platform is realized by executing the kernel functions in parallel through thousands of threads. The inner product of vectors is executed parallelly for all elements by two kernels are shown as Fig. 2.

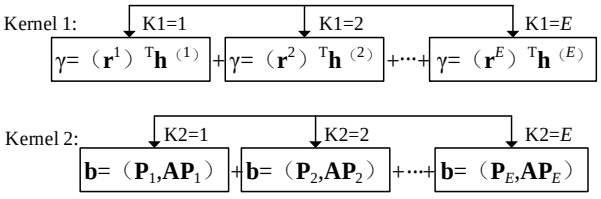

Fig. 2. Kernel functions with reduction operations.

The iterative process of EBE-J-PCG method can be given as:

(1) Initialization

(a) Set initial value,

$$\mathbf{x}_0^{(e)} = 0, \mathbf{r}^{(e)} = \mathbf{b}^e - \mathbf{A}^e\mathbf{x}_0^{(e)}. \tag{13}$$

(b) Jacobi precondition,

$$\mathbf{m}^e = diag\left(\mathbf{A}^e\right), \tag{14}$$

$$\mathbf{m}^{(e)} = \mathbf{m}^{e} \oplus \sum_{j=adj(e)} \mathbf{m}^{j}. \quad (15)$$

(c) Solve equations,

$$\mathbf{m}^{(e)}\mathbf{h}^{e} = \mathbf{r}^{e}. \quad (16)$$

(d) Calculate $\gamma_0 = (\mathbf{r}, \mathbf{h})$.

For e∈(1,2,...,E), use kernel 1 shown as Fig. 2.

(e) Calculate $\mathbf{p}^{(e)}$,

$$\mathbf{p}^{(e)} = \mathbf{h}^{(e)}. \quad (17)$$

(2) Calculate α.

For e∈(1,2,...,E), use kernel 1 and kernel 2 shown as Fig. 2,

$$\alpha = \frac{(\mathbf{P}_{k-1}, \mathbf{h}_{k-1})}{(\mathbf{P}_k, \mathbf{A}\mathbf{P}_k)}. \quad (18)$$

(3) Update $\mathbf{x}^{(e)}$ and $\mathbf{r}^{e}$,

$$\mathbf{x}^{(e)} = \mathbf{x}^{(e)} + \alpha\mathbf{p}^{(e)}, \quad (19)$$

$$\mathbf{r}^{e} = \mathbf{r}^{e} - \alpha\mathbf{A}^{e}\mathbf{p}^{(e)}. \quad (20)$$

(4) Solve equations,

$$\mathbf{m}^{(e)}\mathbf{h}^{e} = \mathbf{r}^{e}. \quad (21)$$

(5) Calculate $\gamma_{new} = (\mathbf{r}, \mathbf{h})$.

For e∈(1,2,...,E), use kernel 1 shown as Fig. 2.

(6) Judge convergence.

If $\gamma_{new} < \delta\gamma_0$, the calculation stops, else,

update $\mathbf{p}^{(e)}$,

$$\mathbf{p}^{(e)} = \mathbf{h}^{(e)} + (\gamma_{new} / \gamma_0)\mathbf{p}^{(e)}, \gamma_0 = \gamma_{new}. \quad (22)$$

Then, return to (2), where he is the element preprocessing vector, me is the vector of the main diagonal elements of each element.

Both of two kernels shown in Fig. 2 involve reduction operation. In EBE-J-PCG method, lots of reduction operations generate a large number of thread branches, which eventually lead to a deterioration in parallel efficiency. Although thousands of threads execute the same kernel functions, their operations and processed data may be different, which is due to the different allocation methods of threads.

C. Branch optimization method of thread-data remapping

Generally, threads are executed in parallel, however, if different threads contain different control conditions, threads that execute different condition paths can only be executed serially. This is called thread branch, which is one of the main factors affecting GPU parallel efficiency.

Path vector is introduced to express thread branches in a piece of code. **V** represents all possible thread path sets, take a warp containing 4 threads as an example:

$$\mathbf{V}[\text{tid}] = \{p_m[\text{tid}], p_n[\text{tid}], p_k[\text{tid}], p_l[\text{tid}]\}, \quad (23)$$

where p[tid] is the execution path, tid is the address of the current thread, and *m*, *n*, *k*, *l* represent four possible different paths respectively. When there is only one element in **V**, such as $\mathbf{V}[\text{tid}] = \{p_m[\text{tid}]\}$, thread branches do not occur in a warp.

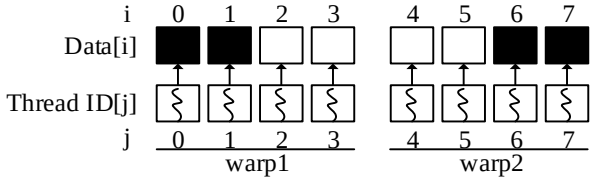

Fig. 3. Original thread-data mapping.

Suppose there are four threads in a warp shown as Fig. 3, squares in two different colors represent the data to be processed differently. The mapping relationship in warp1 is:

$$\text{Thread ID}[j](j = 0,1) \rightarrow \text{Data1}[i](i = 0,1), \quad (24)$$

$$\text{Thread ID}[j](j = 2,3) \rightarrow \text{Data2}[i](i = 2,3). \quad (25)$$

The mapping relationship in warp2 is:

$$\text{Thread ID}[j](j = 4,5) \rightarrow \text{Data2}[i](i = 4,5), \quad (26)$$

$$\text{Thread ID}[j](j = 6,7) \rightarrow \text{Data1}[i](i = 6,7). \quad (27)$$

A mapping relationship represents an execution path. The path vector is $\mathbf{V}[\text{tid}] = \{p_1[\text{tid}], p_2[\text{tid}]\}$ for both warp1 and warp2. Two execution paths exist in both warp1 and warp2, which results in thread branches.

To eliminate the threads branch, the mapping between thread and data can be reset, switch threads that execute the same code in different warp to the same warp so that all threads in a warp will take the same path.

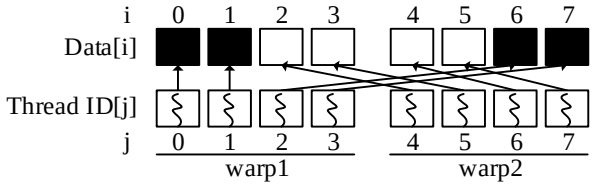

Fig. 4. Redirected thread-data mapping.

As shown in Fig. 4, change the direction of thread mapping, remap the threads in warp1 and warp2 so that the same type of data will be processed in the same warp. After remapping, the mapping relationship in warp1 is:

$$\text{Thread ID}[j](j = 0,1,2,3) \rightarrow \text{Data1}[i](i = 0,1,6,7). \quad (28)$$

The mapping relationship in warp2 is:

$$\text{Thread ID}[j](j = 4,5,6,7) \rightarrow \text{Data2}[i](i = 2,3,4,5), \quad (29)$$

therefore, path vector has been converted to $\mathbf{V}[\text{tid}] = \{p_1[\text{tid}]\}$ for warp1 and $\mathbf{V}[\text{tid}] = \{p_2[\text{tid}]\}$ for

warp2, no thread branches exists in them.

D. Branch optimization of reduction operation in EBE-FEM

Generally, reduction operation adopts adjacent addressing mode, but this mode will generate many thread branches when EBE-FEM is executed on CUDA platform.

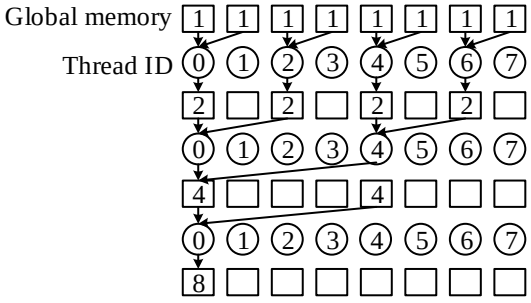

Fig. 5. Adjacent addressing mode.

Figure 5 shows the thread-data mapping of reduction program using adjacent addressing (Assume the reduction operation generates the sum of eight input data). Threads with even address execute addition operation for two adjacent data, threads with odd address do nothing except the parity checking of thread address. Even if only one thread involved in reduction operation, it needs to wait for the other seven threads to perform parity.

All threads in the warp need to be determined the parity, which will generate lots of thread branches. Thread-data remapping method is applied to addressing process to obtain better allocation of threads. Figure 6 shows the optimized addressing mode.

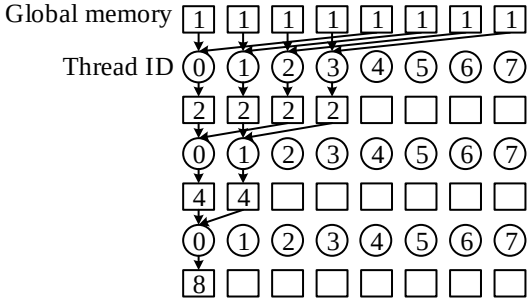

Fig. 6. Optimized addressing mode.

Change the thread data mapping relationship of adjacent addressing, the data involved in reduction operation are divided into two groups on average. The threads that execute addition operation are mapped sequentially to two data from the first group and the second group respectively. The threads in the second group are not involved in any operation, so they can be stopped by hardware, which will reduce useless operations and waiting. After thread-data remapping, the judgments for parity have been eliminated, thread branches have been reduced, which would improve the computational efficiency.

Branch optimization is an optimization method to GPU parallel technology, which improves parallel efficiency by reducing the number of logical judgments. In a multi-GPU environment, the degree of parallelism is higher and branch optimization is more critical, which can improve parallel efficiency more effectively.

III. APPLICATION AND ANALYSIS

The proposed method has been applied to analyze the magnetic field in an opening slot of motor and the field produced by a single-phase power transformer. All the programs have been developed in C++. The computations have been carried out on the Intel Xeon E5-2650 v2, 2.6GHz server with dual GPU (Nvidia Quadro K2000) and 256GB memory.

The main computation in this paper is the inner product of vectors, which is not complicated, and a single GPU can meet the computing requirements. In the subsequent research of multiphysics coupling problems, if the kernel function involves intensive complex parallel calculations, the multi-GPU method will be considered.

A. Magnetic field in an opening slot of motor

The opening slot of motor is shown as Fig. 7, it is an example to illustrate the traditional FEM in a book [11]. Due to its very simple model, few meshes and very regular mesh shapes, the magnetic potential on each node can be determined accurately by traditional FEM, and the results are given by the book.

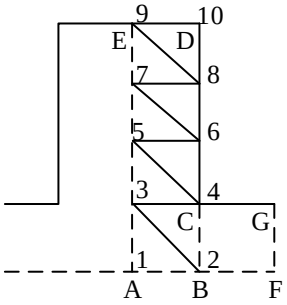

Fig. 7. Model and mesh of opening slot of motor.

In the model, AE and FG are the center lines of stator slot and tooth respectively, and AF is the center line of the air gap. Assuming BC is a magnetic line, rectangular ABDE can be taken as the solving region. Taking scalar magnetic potential φ_m as variable, the first boundary condition exists on AB and CDE. Setting $\varphi_m = 0$ on CDE, and $\varphi_m = 700$ A on AB, the magnetic potential on each node can be obtained.

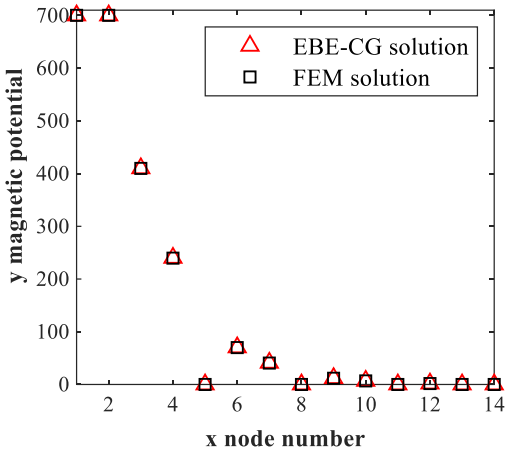

Fig. 8. Comparison of calculation results.

Figure 8 shows the comparison of the results calculated by parallel program of EBE-J-PCG method and traditional FEM. The maximum local error is 0.83%, which verifies the correctness of the EBE-J-PCG method and program.

B. Magnetic field analysis of a single-phase transformer

This method is applied to calculate the 2D quasi-static magnetic field distribution of single-phase DSP-241000kVA/500kV transformer which the secondary side is opened and the primary side is excited by rated current. Figure 9 shows the model and meshes of the transformer. The distribution of the magnetic lines and the magnetic flux density are shown in Figs. 10 and 11.

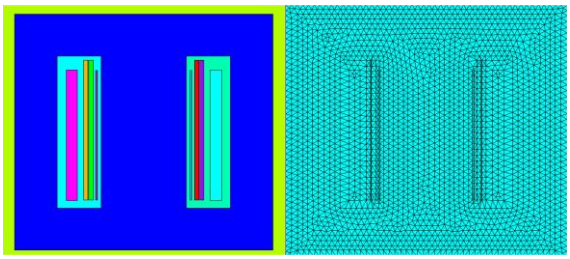

Fig. 9. The model and mesh of the transformer.

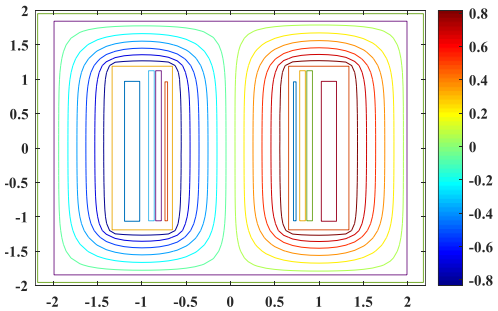

Fig. 10. Distribution of magnetic lines in transformer.

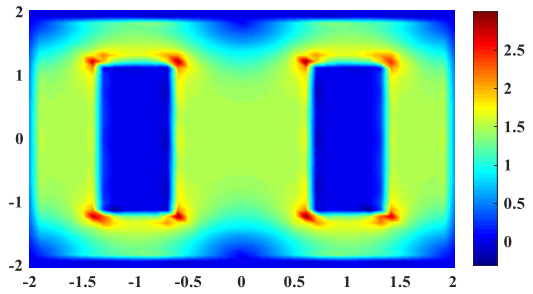

Fig. 11. Distribution of magnetic flux-density in transformer.

To investigate the effect of branch optimization, the transformer model has been divided into different mesh size. Furthermore, the magnetic field in the transformer is calculated by serial EBE-J-PCG, unoptimized parallel EBE-J-PCG and optimized parallel EBE-J-PCG respectively for comparison. The serial EBE-J-PCG program runs on CPU only, while the two parallel programs run on CUDA platform. The computation accuracy of the three methods are the same under the same mesh. Table 1 shows the calculation time of serial EBE-J-PCG program and speedups of parallel EBE-J-PCG program in different mesh size. Speedup is obtained by dividing serial time by parallel time (All the time in the table is in minutes).

Table 1: Calculation results of transformer model

Elements	Serial Time	Parallel without Optimization		Parallel with Optimization	
		Time	Speedup	Time	Speedup
8732	1.5	0.7	2.1	0.4	3.8
11443	2.6	1.2	2.2	0.6	4.3
40816	45.2	18.5	2.4	10.2	4.4
171338	1541.3	593.7	2.6	174.5	8.8

It can be seen from Table 1, compared with serial EBE-J-PCG method running on CPU only, parallel implementation on CUDA platform can improve the computation efficiency. However, thread branches in parallel computing can degrade computational efficiency, which can be seen from the similarity of speedup ratios obtained by non-optimized programs in computing different number of mesh. The proposed thread-data remapping method can solve this problem, the speedup ratios of parallel computation are improved obviously after optimization. Moreover, the larger scale computation is involved, the better acceleration can be obtained.

VI. CONCLUSION

In this paper, the EBE-J-PCG method has been implemented in parallel on CUDA platform, and thread branche in GPU kernel has been researched. The

proposed thread-data remapping method can solve the problem of deterioration in parallel efficiency caused by thread branches, and this optimization method can improve the speedup ratio more obviously. Except the reduction operations, any branching parts of parallel programs based on CUDA platform can adopt the branch optimization method to improve parallel efficiency.

ACKNOWLEDGMENT

This work is supported in part by the National Natural Science Foundation under Grant 51577122 and in part by the Natural Science Foundation of Liaoning Province under Grant 20180550860.

REFERENCES

[1] D. M. Fernandez, M. M. Dehnavi, W. J. Gross, and D. Giannacopoulos, "Alternate parallel processing approach for FEM," *IEEE Trans. Magn.*, vol. 48, no. 2, pp. 399-402, Feb. 2012.

[2] E. Barragy, G. Raham, and F. Carey, "Parallel element-by-element solution scheme," *Int. J. Numer. Methods Eng.*, vol. 26, no. 11, pp. 2367-2382, 1988.

[3] X. Yan, X. Han, D. Wu, D. Xie, B. Bai, and Z. Ren, "Research on preconditioned conjugate gradient method based on EBE-FEM and the application in electromagnetic field analysis," *IEEE Trans. Magn.*, vol. 53, no. 6, pp. 1-4, June 2017.

[4] A. Capozzoli, O. Kilic, C. Curcio, and A. Liseno, "The success of GPU computing in applied electromagnetics," *Applied Comp. Electromag. Soc. Journal*, vol. 33, no. 2, pp. 148-151, Feb. 2018.

[5] J. C. K. Wake and S. Watanabe, "Scalable GPU-parallelized FDTD method for analysis of large-scale electromagnetic dosimetry problems," *Applied Comp. Electromag. Soc. Journal*, vol. 31, no. 6, pp. 661-668, June 2016.

[6] D. Wu, X. Yan, R. Tang, and D. Xie, "Parallel realization of element by element analysis of eddy current field based on graphic processing unit," *Applied Comp. Electromag. Soc. Journal*, vol. 33, no. 2, pp. 168-171, Feb. 2018.

[7] J. Nickolls, "GPU parallel computing architecture and CUDA programming model," *IEEE Hot Chips Symp., HCS*, pp. 1-12, May 2007.

[8] I. Kiss, S. Gyimothy, Z. Badics and J. Pavo, "Parallel realization of the element-by-element FEM technique by CUDA," *IEEE Trans. Magn.*, vol. 48, no. 2, pp. 507-510, Feb. 2012.

[9] S. Wang, X. Yan, Y. Zhang, and D. Wu, "Research on EBE-FEM realized by CUDA applying to electromagnetic field analysis," *IEEE Stud. Conf. Electric Mach. Syst., SCEMS*, 2018.

[10] E. Zhang, Y. Jiang, and Z. Guo, "Streamlining GPU applications on the fly: Thread branch elimination through runtime thread-data remapping," *Proc. Int. Conf. Supercomputing*, pp. 115-125, 2010.

[11] Z. Hu, *Analysis and Calculation of Motor Electromagnetic Fields*. Beijing: China Machine Press, 1989.

Yan Zhang He received his B.S. degree in Electrical Engineering and Automation from Langfang Normal University in 2017. He is currently working towards M.S degree in Shenyang University of Technology. His research interests include numerical analysis and parallel calculation of electromagnetic fields on CUDA platform.

Xiuke Yan received her B.S. degree, M.S. degree and Ph.D. degree in Electrical Engineering from Shenyang University of Technology, China, in 1996, 1999 and 2005, respectively. She is currently a Professor in Shenyang University of Technology. Her research interests include numerical analysis of coupled field and optimization design of electrical equipment, parallel algorithm research of finite element method.