

M2L optimization in FMBEM and its GPU implementation

Zhaohui Xia, Qifu Wang, Yunhua Liu, Yingjun Wang
& Yixiong Wei

*National CAD Support Software Engineering Research Center,
Huazhong University of Science and Technology, China*

Abstract

The translation from multipole moments to local moments (M2L) in the fast multipole boundary element method (FMBEM) costs too much time; we compare three methods of M2L optimization from the three following aspects: accuracy, efficiency and memory usage with an engineering numerical example, and then present a GPU parallel algorithm using CUDA for one of the front three methods which transfers child cell's coefficients to their father cell, meanwhile, improve the tree structure by redefining the whole cells in different levels which can avoid writing data conflict in the parallel strategy. Finally, we use the three-dimensional elastic BEM problems of chassis parts to verify the algorithm, and the result shows that the accelerating effect of this method is significant.

Keywords: fast multipole method, boundary element method, 3D elasticity, GPU, CUDA.

1 Introduction

The fast multipole method (FMM) was invented in the late 1980s by Rokhlin and Greengard [2]. The basic idea of the FMM is to unite as a collection of particles by region, and then replace the effect between the two particles with the particle collection which should adopt the hierarchical tree data to reduce the complexity of N-body problems [1]. There are two types of the tree structures in the FMM: the full tree structure [2, 3] and the adaptive tree structure [4, 5], but nowadays the adaptive tree structure is widely used for its high efficiency and adaptivity.

The boundary element method (BEM) combined with the FMM can efficiently deal with large-scale engineering and scientific problems [6, 7], which



is called fast multipole BEM (FMBEM). According to the time-consuming analysis of the various parts of the FMBEM, it was found that the translation from multipole moments to local moments (M2L) is one of the most time-consuming parts, if M2L calculation could be accelerated, the efficiency of the fast multipole algorithm would be improved in whole.

In order to accelerate M2L calculation, Greengard and Rokhlin [3] introduced the exponential moments to the M2L process in the late 1990s. They proposed a new type of fast multipole algorithm whose precision is controlled and acceleration effect is remarkable when the order of the moments is large, but this algorithm costs large memory and its acceleration effect is not obvious if the order of the moments is small. In recent years, Gumerov and Duraiswami [8] optimized the M2L process of the FMM, the multipole moment coefficients were transferred to the local moment coefficients of the target cell's father cell when the cell meets the condition of a certain distance; thereby the amount of computing M2L is reduced. The corresponding research about this is very little because the optimization is relatively new. Furthermore, Bapat and Liu [9] proposed a similar M2L optimization method which transfers the multipole moment coefficients of a father cell to the local moment coefficient of the target cell directly [10].

In this paper, we completed a GPU parallel algorithm for M2L optimization using CUDA, and then this paper is organized as follows: In Section 2, we analyze and compare three M2L optimization methods through a numerical example from three aspects: accuracy, efficiency and memory. In Section 3, a GPU parallel strategy on the three-dimensional elastic problems is presented and the GPU algorithm is implemented. In Section 4, the example and results are discussed in detail. Finally, the paper is summarized, the presented method is discussed and the results are obtained in Section 5.

2 Method of M2L optimization

2.1 Basic M2L theory

In the fast multipole algorithm, the local moment coefficient is obtained from the transmission from multipole moments to local moments [11]. As shown in Fig. 1, the local moment coefficients get from multipole moments coefficient when it meets the condition of $|\overline{Q_c P_c}| > (|\overline{Q_c Q}| + |\overline{P_c P}|)$, the formula is as follows:

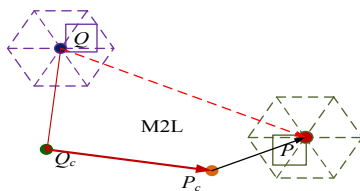


Figure 1: Translation from the multipole moment to local moment.

In order to facilitate operating the data in the tree structure of the Fast Multipole Boundary Element Method (FMBEM), List 2 recorded as $L_2(c)$, also known as interaction list of c , is the collections which contain the cells that are not adjacent with c but their parent adjacent with the c 's parent.

2.2 Optimization methods for M2L

M2L optimization method based on the exponential moments with the introduction of this new concept, makes M2L process replaced by the coefficient transformation of multipole moments to exponential moments (M2X for short), the coefficient transmission between the exponential expansions (X2X for short) and the coefficient transformation of exponential moments to local moments (X2L for short) processes [3]. The Fast Multipole Boundary Element Method (FMBEM) with exponential moments was called New FMBEM [12]. Its process is shown in Figs 2 and 3.

The transmission of the previous M2L is used among the cells in the same level, each cell contains a maximum of 189 cells which meet the M2L transmission requirement. When two cells are not in the same level, however, they may still meet the distance condition which can be operated M2L transmission. This transmission method from child cells to father cell makes the maximum M2L times down to 119 and improve the computational efficiency under the condition of meeting the original error limits [8].

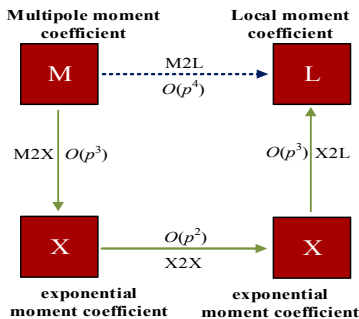


Figure 2: Converting from M2L to M2X, X2X and X2L.

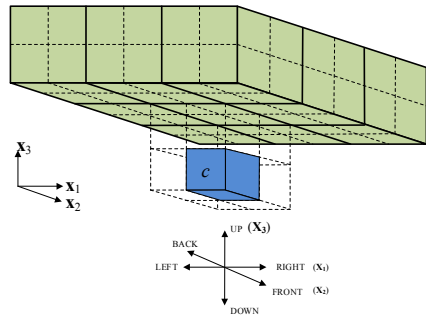


Figure 3: Cube c , and the cube that meet the moment condition.

The principles of this method are shown as follows. $L_2(c)$ can be subdivided to $L_{2,1}(c)$ and $L_{2,2}(c)$ according to the distance between the cell and c 's parent cell, among this, $L_{2,1}(c)$ represents the part collections of the far away and $L_{2,2}(c)$ is the close part (shown as Fig. 4). Assume that the side length of the cube represented by c is l , the distance between the centre cell of $L_2(c)$ and c 's parent cell is r , so there exists the following relationship for any cell in the collection $L_2(c)$ in the formula (1):

$$\begin{cases} \frac{\sqrt{3}}{2}l \leq r \leq \frac{\sqrt{35}}{2}l, d \in L_{2,1}(c) \\ \frac{\sqrt{43}}{2}l \leq r \leq \frac{5\sqrt{3}}{2}l, d \in L_{2,2}(c) \end{cases} \quad (1)$$

$$\begin{cases} \frac{\sqrt{3}}{2} \leq r \leq 2l \\ 2l < r \leq \frac{\sqrt{106}}{4}l \end{cases} \quad (2)$$

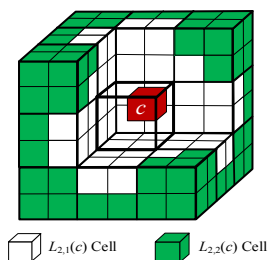


Figure 4: $L_2(c)$ -classification based on the distance of c 's father cell.

This optimization method is very similar with the method in Section 2.2 except its translation direction and list subdivision. In this method, $L_2(c)$ can be subdivided to $L_{2,1}(c)$ and $L_{2,2}(c)$ according to the distance r , assume that the side length of the father cell is l [9], it is shown as formula (2).

2.3 Analysis of the optimization methods

The programs of this section are executed on a desktop computer: Intel i7-2600K 3.40 GHz (Single-core calculation), OS is Windows XP, RAM is DDR3 SDRAM (2GB), the compiler for CPU code is Microsoft Visual Studio 2008, double precision is used in the CPU numerical example. This example is a chassis part whose size and boundary conditions are shown in Fig. 5. The surface of the chassis part is meshed with linear triangular elements, and the degrees of freedom (DOFs) are 19314, and the order of the multipole moment and local moment is separately set to 6, 8 and 10, the convergence residual error of the GMRES algorithm is set to 10^{-3} .

The comparison of the example accuracy is the right side vector \mathbf{b} of the boundary element system equations $\mathbf{Ax}=\mathbf{b}$. \mathbf{b}_{bem} is the result vector of direct matrix vector multiplication in the traditional BEM, and \mathbf{b}_{fbem} represents the result vector of the different M2L of the FMBEM, the relative error between the two is defined as follows:

$$\varepsilon = \frac{\|\mathbf{b}_{bem} - \mathbf{b}_{fbem}\|_2}{\|\mathbf{b}_{bem}\|_2} \quad (3)$$

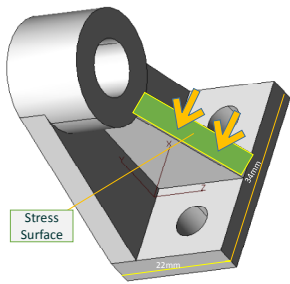


Figure 5: Chassis parts model.

The results and comparisons are shown in Tables 1–3. Through them, we found that exponential moment method’s accuracy is high, but its accuracy depends on both the order of the exponential moment and the moment, so the target accuracy is needed to the reasonable collocation for them. Translation across level method does not need additional memory, but the error of the translation from the father cell to the child cell is relatively large, so the overall effect of the method transferring from the child cells to the father cell is the best which is especially suitable for engineering calculation.

Table 1: Relative error of the right vector b of different M2L methods.

p	M2L	M2L_E		M2L_N	M2L_NF
		$g=8$	$g=17$		
6	1.283×10^{-4}	6.466×10^{-4}	1.284×10^{-4}	2.150×10^{-4}	2.223×10^{-4}
8	1.552×10^{-5}	6.336×10^{-4}	1.582×10^{-5}	2.762×10^{-5}	2.856×10^{-5}
10	2.275×10^{-6}	6.335×10^{-4}	2.208×10^{-6}	5.240×10^{-6}	5.418×10^{-6}

Table 2: Consumption time of different M2L methods (units: CPU-Q9400).

p	M2L	M2L_E		M2L_N	M2L_NF
		$g=8$	$g=17$		
6	14.8	14.9	15.85	14.7	14.8
8	30.6	28.8	51.2	25.6	25.9
10	53.7	29.9	67.7	41.8	41.9

Table 3: The amount of memory of different M2L methods (unit: MB)

p	M2L	M2L_E		M2L_N	M2L_NF
		$g=8$	$g=17$		
6	114.1	115.9	119.3	114.1	114.1
8	114.7	135.6	139.3	114.7	114.7
10	115.5	149.0	154.1	115.5	115.5

3 Implementation of parallel strategy using CUDA

3.1 GPU parallel computing based CUDA architecture

CUDA, based on the C language and can write the execution program of the display chip through the language similar with C language, is NVIDIA GPU (General Purpose GPU) model [13], and its computation is based on GPU. In the CUDA architecture, the program consists of the host and the device two parts [8], among them, the host is responsible for the serial parts while the device for the parallel [14]. The program of the device is known as “Kernel”. The smallest execution unit of GPU is Thread, and the CUDA architecture can contain tens of thousands of thread which is very suitable for large-scale fine-grained data parallel computation [15, 16].

3.2 GPU parallel strategy of M2L

According to the theory of the chapter 2.1, the outermost level cycle is cell circulation rather than level circulation which shows that M2L is very suitable for parallel computation. In the parallel strategy one Block is responsible for a M2L transmission of each cell, and one thread is responsible for the computation of the coefficient which is corresponding to a set of subscript (n, m) of the formula (2.1).

When using the M2L transmission method of section 2.2, algorithm 1 of Section 2.1 can't be paralleled directly because of writing data conflict problem. Shown as Fig. 7, cell c_1 and c_2 have the same father cell and cell d_1 , cell d_1 and cell d_2 respectively belong to $L_{2,2}(c_1)$ and $L_{2,2}(c_2)$ which meet the requirement of M2L transmission across the level. If their multipole moments coefficients were transferred to the center of the c_1 and c_2 's father cell p_c , there would be writing data conflict shown as Fig. 6, because the local moments coefficient of the c_1 and c_2 's father cell would be modified when the parallel computation is operating.

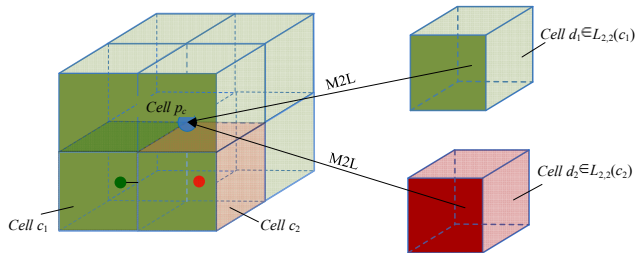


Figure 6: Writing data conflict in M2L method.

In order to overcome this problem, the tree structure should be improved, the list $L_{2,2}(p_c)$ of each cell p_c from the 1st to the last level $nlevel$ should be redefined as $L'_{2,2}(p_c)$ which is the collections of $L_{2,2}(c_i)$ of each child cell of p_c defined in the section 2.2, that is:

$$L'_{2,2}(p_c) = L_{2,2}(c_1) \cup L_{2,2}(c_2) \cup L_{2,2}(c_3) \cup L_{2,2}(c_i) \quad (4)$$

Through the above improvement, there is no writing data conflict in the specific computation, and the computation can be parallel, meanwhile, the M2L computation of the cells from the 1st level should be added while its computation is too little to be ignored because of the rare cells in the 1^{st} level.

Each M2L of the subscript (j, k) is independent, thus “one-block-one-cell” mode of parallelization is adopt to compute M2L shown in Fig. 7. According to the property of the FMBEM, we propose a particular thread distribution strategy that assigns each thread in a block to compute the values of $L_{j,k}(Q_c)$, $L_{n,j,k}(Q_c)$, $L'_{j,k}(Q_c)$ and $L'_{n,j,k}(Q_c)$ of each j and k pair (named (j,k) for short) in Eq. (1), and one Block is responsible for the computation of a target cell shown in Fig. 8. The number of moments order is t , the value domain of (j, k) is

$$\begin{cases} 0 \leq j \leq t \\ -j \leq k \leq j \end{cases} (j, k \in \text{the set of integers}) \quad (5)$$

where it can be obtained that k is symmetrical about 0, and the multipole moments in Eq. (1) have the following property:

$$\begin{aligned} L_{j,-k}(Q_c) &= (-1)^k \overline{L_{j,k}(Q_c)} (k \geq 0), \\ L_{n,j,-k}(Q_c) &= (-1)^k \overline{L_{n,j,k}(Q_c)} (k \geq 0), \\ L'_{j,-k}(Q_c) &= (-1)^k \overline{L'_{j,k}(Q_c)} (k \geq 0), \\ L'_{n,j,-k}(Q_c) &= (-1)^k \overline{L'_{n,j,k}(Q_c)} (k \geq 0) \end{aligned} \quad (6)$$

In order to save the memory which is especially limited in GPU, only the multipole moments of $k \geq 0$ are stored, and the values when $k < 0$ are obtained from Eq. (1). The detail (j,k) distribution is shown in Fig. 9, like the indices of a lower triangular matrix. One dimension arrays are used to store multipole moments, $L_{j,k}(Q_c)$ and $L'_{j,k}(Q_c)$ need one array separately, and $L_{n,j,k}(Q_c)$, $L'_{n,j,k}(Q_c)$ need three arrays respectively (each n of $L_{n,j,k}(Q_c)$ or $L'_{n,j,k}(Q_c)$ need one array). The location of (j,k) in an array is

$$\text{loct}_{j,k} = \frac{j(j+1)}{2} + k \quad (7)$$

as shown in Fig. 8.

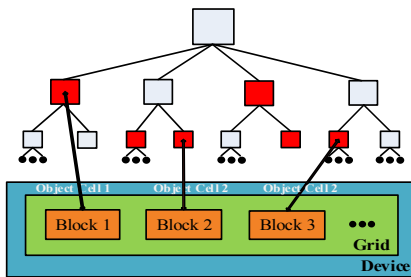


Figure 7: Corresponding relation.

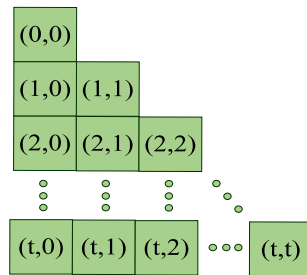


Figure 8: Distribution of the subscript (j, k) between the target cell and the block coefficients needed to calculate.

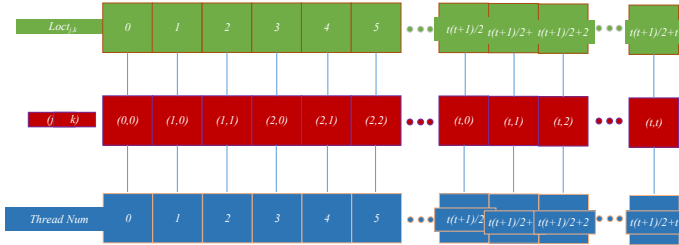


Figure 9: Allocation strategy in the thread of the Block.

One thread is assigned to charge one (j, k) computation of the multipole moments, then the number of threads in one block is $(t+1)(t+2)/2$, and the thread to (j, k) mapping relationship is shown in Fig. 9.

In this parallel mode, all M2L translations are stored in shared memory in the process of parallel computing, and the results are copied from shared memory to global memory. The multipole moments just need to be computed in their own cell, but when the M2L is computed in a cell, all M2L translations of all cells in the interacting cell list of that cell have to be computed and added together. When the number of target cells is large enough, the computing capability of GPU can be fully utilized. In the CUDA architecture, the GPU parallel computing algorithm steps of M2L are as follows:

Algorithm 2: GPU parallel computing algorithm of M2L

Note: $nlevel$ is the tree structure levels, C_i is the i th level of the tree structure.

$N1$ is the sum of the cells from C_1 level to C_{nlevel} , the corresponding Block number is $numblocks1=N1$

$N2$ is the sum of the cells from C_2 level to C_{nlevel} , the corresponding Block number is $numblocks2=N2$

Number of threads in a Block is: $t(t+1)/2+t+1$.

Let bid =block ID, represent the C_{bid} cell, tid =local thread ID in a Block.

(1) and (2) need to call the kernel function respectively

Start algorithm

(1) **for** $bid=0, 1, \dots, numblocks2-1$ **in parallel do**

for cell $d \in L_{2,1}(C_{bid})$ **do**

Each thread tid do parallel computing, transfer from the multipole moments coefficient of the cell d 's subscript (j, k) to the local moments coefficient of the cell C_{bid} 's location, and then do accumulating.

end for d

end for bid

(2) **for** $bid=0, 1, \dots, numblocks2-1$ **in parallel do**

for cell $d \in L_{2,2}(C_{bid})$ **do**

Each thread tid do parallel computing, transfer from the multipole moments coefficient of the cell d 's subscript (j, k) to the local moments coefficient of the cell C_{bid} 's location, and then do accumulating.

end for d

end for bid

End algorithm



4 Numerical example

The FMBEM programs are executed on a desktop computer: Intel Core i7 2600k (3.4 GHz), GPU is NVIDIA GeForce GTX580, OS is Windows 7, RAM is DDR3 SDRAM (8 GB), the compiler for CPU code is Microsoft Visual Studio 2008, and the compiler for GPU codes is NVIDIA CUDA 4.0 (C language), double precision is used in the CPU numerical example while single precision used in the GPU computation. The maximum number of the cells in the leaf cell is set to 32, the order of the moment is set to 6, the residual error for convergence of the GMRES algorithm is set to 10^{-3} , and the finite element is computed by using ANSYS 13.0 software.

This example is the same as which size and boundary conditions are shown in Fig. 2. The Elastic Modulus is 260000 MPa, Possion's ratio is 0.3. The surface of the chassis parts is meshed with linear triangular elements, and the degrees of freedom (DOFs) have three types which are shown in Table 4, and the chassis parts were discreted with tetrahedron quadratic elements in ANSYS whose size is set to the same as the size of the triangular element using in the fast multipole boundary element method.

Table 4 shows the information of the chassis parts model in the different methods to calculate, among them, "FBEM_C" represents the solving in the CPU serial FMBEM, "FBEM_G" is GPU parallel FMBEM to solve. In the ANSYS column, two numerical values are separated by the symbol "||", the former represents the total memory taken up by the solver when solving, and the latter shows that the total memory need by the computing using the in-core way. Obtained from the calculation time, FBEM_C solving speed has exceeded ANSYS, but little difference between the computing time, and FBEM_G computation time significantly smaller than the other two.

Table 4: The related information of different methods for solving machine tool model.

Element size type	DOFs	Iteration number	Total time(s)	Total memory(MB)
FBEM_C(Type 1)	19314	12	14.7	90.3
FBEM_C(Type 2)	37698	45	123.1	230.4
FBEM_C(Type 3)	100029	58	384.6	481.2
FBEM_G(Type 1)	19314	12	7.1	88.6
FBEM_G(Type 2)	37698	43	34.6	180.2
FBEM_G(Type 3)	100029	58	99.2	290.8
ANSYS(Type 1)	137805	-	12.3	848 807
ANSYS(Type 2)	355014	-	253.7	265.5 2841
ANSYS(Type 3)	1726773	-	3706.5	2380 26172

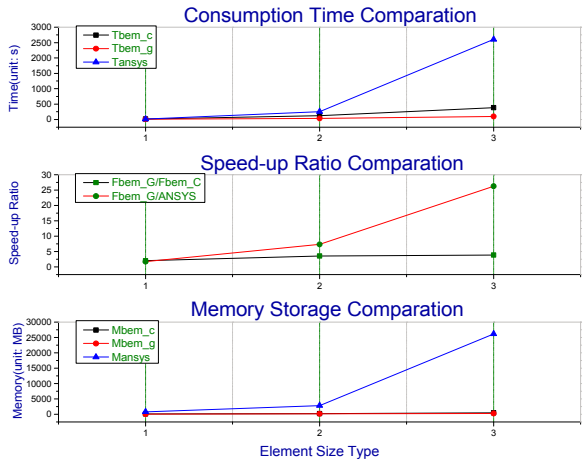


Figure 10: Comparison between Fbem_g, Fbem_c and ANSYS.

At the same time, FBEM_G/ANSYS speedup is larger than FBEM_G/FBEM_C because FBEM_C (memory) and FBEM_G (video memory) are much lower than ANSYS (memory) from the amount of memory taken by the solution. There are three types of element size in the example and the results are shown in Table 4, and the comparison for the consumption time, speed-up ratio and memory usage between the CPU serial, GPU parallel and ANSYS computation are shown in Fig. 10. It can be obtained that the speedup ratio of FBEM_G/FBEM_C and FBEM_G/ANSYS is more and more significant as the computing scale is larger and larger.

The FBEM_C, FBEM_G and ANSYS total displacement and Von Mises Stress of the chassis parts model were given in Figs 11 and 12 The linear element was used in the FBEM_C and FBEM_G computing while the quadratic element in the ANSYS computing. From the figures, it can be observed that the

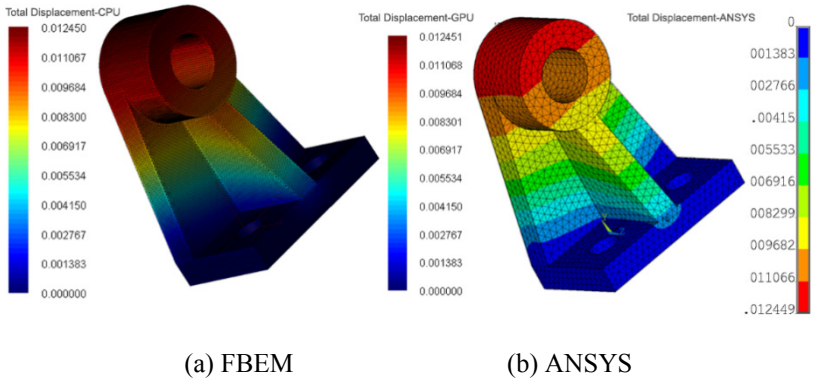


Figure 11: Total displacement distribution of the chassis parts model.

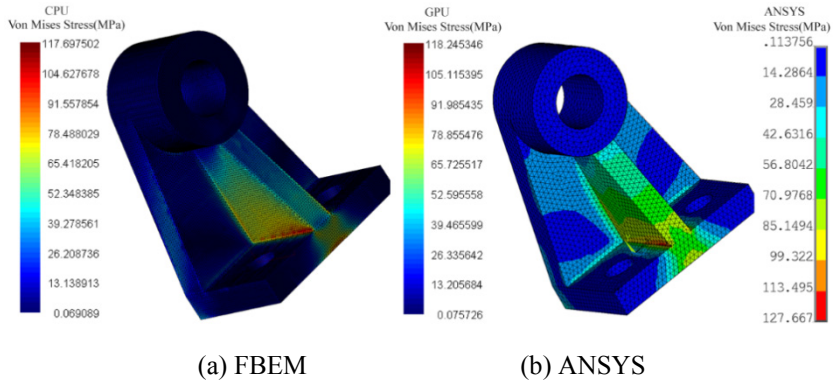


Figure 12: Von Mises Stress distribution of the chassis parts model.

distribution and the results of the three are consistent, the stress error is larger than the displacement; this phenomenon is due to that the stress computation accuracy of the linear element is poorer.

5 Conclusion

We compared the three M2L optimization algorithms of the Fast Multipole Boundary Element Method through an engineering example, and then presented a GPU parallel strategy for one of the three which transferred from the child cell coefficient to the father cell coefficient in the M2L process, and used this GPU acceleration algorithm in a 3D elasticity problem. In order to eliminate the data conflict, the tree structure was improved, and this algorithm was verified by an engineering example. The example results shows that the GPU parallel computing using CUDA in this paper has a significant acceleration effect and less memory usage compared to ANSYS computation under the same scale, and the speed-up ratio of the GPU/ANSYS and GPU/CPU is larger and larger while the computation scale increasing. The experiment results show that the theory and method of this paper is effective and feasible, and the GPU parallel algorithm of the FMBEM can realize the calculation of a larger problem.

In the future, we will do further research on improving the calculation accuracy using quadratic element, optimizing the GPU parallel algorithm in this paper to speed it up further.

Acknowledgements

This research has been supported by the National Natural Science Foundation of China under Grant number 51375185 and the National 863 Project of China under Grant number SS2013AA041301.

References

- [1] Warren, M.S. and J.K. Salmon. A parallel hashed oct-tree n-body algorithm. *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*. 1993. ACM.
- [2] Greengard, L. and V. Rokhlin, A fast algorithm for particle simulations. *Journal of Computational Physics*. **73**(2): pp. 325–348, 1987.
- [3] Greengard, L. and V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta numerica*. **6**(1): pp. 229–269, 1997.
- [4] Cheng, H., L. Greengard, and V. Rokhlin, A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*. **155**(2): pp. 468–498, 1999.
- [5] Zhang, J. and M. Tanaka, Adaptive spatial decomposition in fast multipole method. *Journal of Computational Physics*. **226**(1): pp. 17–28, 2007.
- [6] Wang, Y., *et al.*, An adaptive dual-information FMBEM for 3D elasticity and its GPU implementation. *Engineering Analysis with Boundary Elements*. **37**(2): pp. 236–249, 2013.
- [7] Zhang, J., M. Tanaka, and M. Endo, The hybrid boundary node method accelerated by fast multipole expansion technique for 3D potential problems. *International journal for numerical methods in engineering*. **63**(5): pp. 660–680, 2005.
- [8] Gumerov, N.A. and R. Duraiswami, Fast multipole methods on graphics processors. *Journal of Computational Physics*. **227**(18): pp. 8290–8313, 2008.
- [9] Bapat, M. and Y. Liu, A new adaptive algorithm for the fast multipole boundary element method. *Computer Modeling in Engineering & Sciences (CMES)*. **58**(2): pp. 161–184, 2010.
- [10] Wei, Y., *et al.*, Optimizations for Elastodynamic Simulation Analysis with FMM-DRBEM and CUDA. *Computer Modeling in Engineering & Sciences (CMES)*. **86**(3): pp. 241–273, 2012.
- [11] Liu, Y., *Fast multipole boundary element method: theory and applications in engineering* 2009: Cambridge university press.
- [12] Yoshida, K.-i., Applications of fast multipole method to boundary integral equation method. *Dept. of Global Environment Eng., Kyoto Univ., Japan*: 2001.
- [13] Shen, L. and Y.J. Liu, An adaptive fast multipole boundary element method for three-dimensional potential problems. *Computational Mechanics*. **39**(6): pp. 681–691, 2007.
- [14] Araújo, F.C.d., E.F. d’Azevedo, and L.J. Gray, Boundary-element parallel-computing algorithm for the microstructural analysis of general composites. *Computers & Structures*. **88**(11): pp. 773–784, 2010.
- [15] Labaki, J., E. Mesquita, and L. Saraiva Ferreira. The BEM on general purpose graphics processing units (GPGPU): a study on three distinct implementations. *Advances in boundary element techniques XI*:



Proceedings of the 11th International Conference, Berlin, Germany. United Kingdom: EC Ltd. 2010.

- [16] Cunha, M., J. Telles, and A. Coutinho, A portable parallel implementation of a boundary element elastostatic code for shared and distributed memory systems. *Advances in Engineering Software*. **35**(7): pp. 453–460, 2004.

